DELFT UNIVERSITY OF TECHNOLOGY

REPORT 10-05

Fast solution of nonsymmetric linear systems on Grid computers using parallel variants of IDR(s)

T. P. Collignon and M. B. van Gijzen

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2010

Fast solution of nonsymmetric linear systems on Grid computers using parallel variants of IDR(s)

T. P. Collignon and M. B. van Gijzen *

February 9, 2010

Abstract

IDR(s) is a family of fast algorithms for iteratively solving large nonsymmetric linear systems [14]. With cluster computing and in particular with Grid computing, the inner product is a bottleneck operation. In this paper, three techniques are combined in order to alleviate this bottleneck. Firstly, the efficient and stable IDR(s) algorithm from [16] is reformulated in such a way that it has a single global synchronisation point per iteration step. Secondly, the so–called test matrix is chosen so that the work, communication, and storage involving this matrix is minimised 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 these three techniques.

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 [14]. The method has attracted considerable attention, e.g., see [11, 6, 12, 7, 8]. For $s = 1$, IDR(s) is mathematically equivalent to the ubiquitous Bi–CGSTAB algorithm [15]. For important types

*Delft Institute of Applied Mathematics, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, the Netherlands. Tel: +31 (0)15 2787 608 Fax: +31 (0)15 2787 209, e-mail: T.P.Collignon@tudelft.nl, M.B.vanGijzen@tudelft.nl

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 synchronisation is a bottleneck operation in such computational environments and to alleviate this bottleneck, three techniques are combined:

(i.) The efficient and numerically stable IDR(s)-*biortho* method from [16] is reformulated in such a way that it has one global synchronisation 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 minimise the total computing time using only problem and machine–dependent parameters;

(iii.) Piecewise sparse column vectors for the test matrix are used to minimise computation, communication, and storage involving this matrix in multi–cluster environments.

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 [10]. 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 new variant.

Techniques for reducing the number of synchronisation points in Krylov subspace methods on parallel computers has been studied by several authors [2, 3, 4, 5, 19, 18, 20]. With the advent of Grid computing, the need for reducing global synchronisations is larger than ever. In addition to presenting an IDR(s) variant with a single global synchronisation point per iteration step, this paper also gives a method for a–priori computation of the optimal s to minimise computing times. Lastly, sparse test vectors are used to reduce the cost of global synchronisation even further, resulting in an 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 [14] also has a single global synchronisation point per iteration step, except when computing a new ω each $s + 1$st step, which requires two global synchronisations. However, the IDR(s)-biortho method exhibits superior numerical stability and has less floating point operations per IDR 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 $Q^{\perp}$. Number of iterations refers to the number of (preconditioned) matrix–vector multiplications and the index n refers to the *iteration* number, while the index j always refers to the jth IDR *cycle*. Note that one IDR cycle consists of $s+1$ iterations.

This paper is organised as follows. In Sect. 2 the three strategies for constructing efficient parallel IDR(s) methods are discussed in detail. Section 3 contains extensive experimental results on the DAS–3, demonstrating the effectiveness of the three strategies. Concluding remarks are given in Sect. 4.

2 An efficient IDR(s) algorithm for Grid computing

The IDR–based methods are new iterative algorithms for solving large nonsymmetric systems and much research is needed on efficient parallelisation on distributed memory computers. The large amount of freedom when deriving IDR(s) algorithms allows for efficient tuning of the numerical algorithm to specific computational environments.

This section is structured in the following way. In Sect. 2.1 the IDR(s)–biortho method is reproduced and used as a basis for the IDR(s)–minsync method. Section 2.2 describes the parallel performance model that is used to predict the optimal value of s and in Sect. 2.3 the method of using sparse column vectors for the test matrix is described.

2.1 An efficient IDR(s) variant with minimal synchronisation points

Given an initial approximation x_0 to the solution, all IDR(s) methods construct residuals $r_j = b - Ax_j$ 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*

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

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

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

Ultimately, the residual is forced in the zero–dimensional subspace $\mathcal{G}_j = \{0\}$ for some $j \leq N$. For a proof the reader is referred to [14, 12].

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

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

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

It can be shown that in exact arithmetic, IDR(s) methods terminate within N/s dimension reduction steps, or equivalently, within $N(1+\frac{1}{s})$ (preconditioned) matrix–vector multiplications [14, §3]. In practical applications, the iteration process will exhibit much faster convergence rates according to $\widehat{N}/s$ IDR cycles, where $\widehat{N} \ll N$.

Shown in Alg. 1 is the (right) preconditioned IDR(s)–biortho variant [16], which not only has slightly less operations but is also numerically more stable than the IDR(s)–proto variant. The dimension reduction step (i.e., lines 31–36) consists of one preconditioned matrix–vector product, two vector updates, and two inner products. Combined with the operations for constructing s vectors in $\mathcal{G}_j$ (i.e., lines 5–30), 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 synchronisation points per IDR 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 vector overhead. Let r_{n+1} be the first residual in $\mathcal{G}_{j+1}$. In IDR(s)-biortho, the s vectors for $\mathcal{G}_{j+1}$ are made to satisfy

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

and the intermediate residuals are made to satisfy

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

In the implementation presented in Alg. 1, these conditions are enforced using a modified Gram–Schmidt (MGS) process for oblique projection (lines 14–23). 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–orthogonalisation process using solely *scalar updates*. For a more detailed discussion on using either CGS or MGS for the oblique projections, see [6].

Shown in Alg. 2 is the reformulated variant IDR(s)-minsync. In the following, the two phases of the new IDR variant are discussed separately, where we sometimes specifically refer to line numbers in Alg. 2.

Algorithm 1 IDR(s) 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 $||b - Ax|| \leq \varepsilon$.

Set $G = U = 0 \in \mathbb{C}^{N\times s}; M = (\mu) = I \in \mathbb{C}^{s\times s}; \omega = 1$
Compute $r = b - Ax$
// loop over nested $\mathcal{G}_j$ spaces, $j = 0, 1, \ldots$
while $||r|| \geq \varepsilon$ **do**
  // Compute s linearly independent vectors g_k in $\mathcal{G}_j$
  $\boxed{\phi = Q^H r}$, $\phi = (\phi_1, \ldots, \phi_s)^T$ *// s inner products (combined)*
  for $k = 1$ to s **do**
    Solve $M\gamma = \phi$ for γ, $\gamma = (\gamma_k, \ldots, \gamma_s)^T$
    $v = r - \sum_{i=k}^{s} \gamma_i g_i$
    $\tilde{v} = B^{-1}v$ *// Preconditioning*
    $u_k = \sum_{i=k}^{s} \gamma_i u_i + \omega\tilde{v}$
    $g_k = Au_k$
    // Make g_k orthogonal to $q_1, \ldots, q_{k-1}$
    for $i = 1$ to $k - 1$ **do**
      $\boxed{\alpha = q_i^H g_k / \mu_{i,i}}$ *// $k - 1$ inner products (separate)*
      $g_k \leftarrow g_k - \alpha g_i$
      $u_k \leftarrow u_k - \alpha u_i$
    end for
    // Update column k of M
    $\boxed{\mu_{i,k} = q_i^H g_k}$ for $i = k, \ldots, s$ *// $s - k + 1$ inner products (combined)*
    // Make the residual orthogonal to $q_1, \ldots, q_k$
    $\beta = \phi_k / \mu_{k,k}$
    $r \leftarrow r - \beta g_k$
    $x \leftarrow x + \beta u_k$
    // Update $\phi = Q^H r$
    if $k + 1 \leq s$ **then**
      $\phi_i = 0$ for $i = 1, \ldots, k$
      $\phi_i = \phi_i - \beta\mu_{i,k}$ for $i = k + 1, \ldots, s$
    end if
  end for
  // Entering $\mathcal{G}_{j+1}$, the dimension reduction step
  $\tilde{v} = B^{-1}r$ *// Preconditioning*
  $t = A\tilde{v}$
  $\boxed{\omega = (t^H r)/(t^H t)}$ *// Two inner products (combined)*
  $r \leftarrow r - \omega t$
  $x \leftarrow x + \omega\tilde{v}$
end while

Algorithm 2 IDR(s) with bi–orthogonalisation of the 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 $||b - Ax|| \leq \varepsilon$.

// Initialisation
$G = U = 0 \in \mathbb{C}^{N\times s}; M = [\mu] = I \in \mathbb{C}^{s\times s}; \omega = 1$
Compute $r = b - Ax$
$\phi = Q^H r, \phi = (\phi_1, \ldots, \phi_s)^T$
// Loop over nested $\mathcal{G}_j$ spaces, $j = 0, 1, \ldots$
while $||r|| > \varepsilon$ **do**
  // Compute s linearly independent vectors g_k in $\mathcal{G}_j$
  for $k = 1$ to s **do**
    // Compute $v \in \mathcal{G}_j \cap Q^\perp$
    Solve $M_l \gamma_{(k:s)} = \phi_{(k:s)}$
    $v = r - \sum_{i=k}^{s} \gamma_i g_i$
    $\tilde{v} = B^{-1} v$ *// Preconditioning step*
    $u_k = \sum_{i=k}^{s} \gamma_i u_i + \omega \tilde{v}$
    $g_k = A u_k$
    $\boxed{\psi_i = q_i^H g_k}$ for $i = 1, \ldots, s$ *// s inner products (combined)*
    Solve $M_t \alpha_{(1:k-1)} = \psi_{(1:k-1)}$
    // Make g_k orthogonal to $q_1, \ldots, q_{k-1}$ and update u_k accordingly
    $g_k \leftarrow g_k - \sum_{i=1}^{k-1} \alpha_i g_i, \quad u_k \leftarrow u_k - \sum_{i=1}^{k-1} \alpha_i u_i$
    // Update column k of M
    $\mu_{i,k} = \psi_i - \sum_{j=1}^{k-1} \alpha_j \mu_{i,j}^c$ for $i = k, \ldots, s$
    // Make r orthogonal to $q_1, \ldots, q_k$
    $\beta = \phi_k / \mu_{k,k}$
    $r \leftarrow r - \beta g_k$
    $x \leftarrow x + \beta u_k$
    // Update $\phi = Q^H r$
    if $k + 1 \leq s$ **then**
      $\phi_i = 0$ for $i = 1, \ldots, k$
      $\phi_i \leftarrow \phi_i - \beta \mu_{i,k}$ for $i = k + 1, \ldots, s$
    end if
  end for
  // Entering $\mathcal{G}_{j+1}$. Note: $r \perp Q$
  $\tilde{v} = B^{-1} r$ *// Preconditioning step*
  $t = A\tilde{v}$
  $\boxed{\omega = (t^H r)/(t^H t); \phi = -Q^H t}$ *// $s + 2$ inner products (combined)*
  $r \leftarrow r - \omega t$
  $x \leftarrow x + \omega \tilde{v}$
  $\phi \leftarrow \omega \phi$
end while

1. The dimension reduction step The orthogonality condition (4) implies that the last intermediate residual before the dimension reduction step is orthogonal to all the columns of Q. Therefore, $r_n \in \mathcal{G}_j \cap Q^\perp$ and according to Theorem 1 the first residual r_{n+1} in $\mathcal{G}_{j+1}$ is thus computed as

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

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

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

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 6 from Alg. 1 as follows. Premultiplying the recursion for computing the new residual (5) with Q^H gives

$$\begin{aligned} Q^H r_{n+1} &= Q^H r_n - \omega_{j+1} Q^H AB^{-1} r_n & (8)\\ &= -\omega_{j+1} Q^H AB^{-1} r_n, & (9) \end{aligned}$$

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).

2. Generating s additional vectors in $\mathcal{G}_{j+1}$ In addition to the standard orthogonalisation step performed in IDR(s) for computing a vector $v \in \mathcal{G}_j \cap Q^\perp$ (line 9–11), 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 orthogonalisations that need to be performed. Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, the two orthogonalisations 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 $g_i = A u_i$ for all i.

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

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

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

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

In the implementation of IDR(s)-biortho from Alg. 1, the vector $\widehat{g}_{n+k}$ is subsequently orthogonalised against $q_1, \ldots, q_{k-1}$ using a MGS process (lines 14–18 in Alg. 1), while v_{n+k} is orthogonalised using a CGS process (lines 8–9 in Alg. 1).

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

$$M_l \equiv [\mu_{i,j}^l] = \begin{cases} q_i^H g_{n+j-s-1} & \text{for } k \le j \le i \le s; \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

and

$$M_t \equiv [\mu_{i,j}^t] = \begin{cases} q_i^H g_{n+j} & \text{for } 1 \le j \le i \le k-1; \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

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

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

$$\psi_i = q_i^H \widehat{g}_{n+k}, \quad \text{for } 1 \le i \le s \qquad \text{(line 15)} \tag{15}$$

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

$$\begin{cases} M_l \gamma_{(k:s)} & = \phi_{(k:s)} \qquad \text{(line 10)} \\ M_t \alpha_{(1:k-1)} & = \psi_{(1:k-1)} \qquad \text{(line 16)} \end{cases} \tag{16}$$

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

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:

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

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

$$M_c \equiv [\mu^c_{i,j}] = q_i^H g_{n+j} \quad \text{for} \quad k \le i \le s, \quad 1 \le j \le k-1 \tag{18}$$

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

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

then

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

Also, 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).

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

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

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

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

To summarise, this gives for step k while referring to the line numbers:

$$\begin{aligned}
\psi_i &= q_i^H \widehat{g}_k, \quad i = 1, \ldots, s && \text{(line 15, } s \text{ combined inner products)} \\
\alpha_{(1:k-1)} &= M_t^{-1} \psi_{(1:k-1)} && \text{(line 16, lower triangular system } M_t^{-1}\text{)} \\
g_{n+k} &= \widehat{g}_{n+k} - \sum_{j=1}^{k-1} \alpha_j g_{n+j} && \text{(line 18, orthogonalise against } q_1, \ldots, q_{k-1}\text{)} \\
u_{n+k} &= \widehat{u}_{n+k} - \sum_{j=1}^{k-1} \alpha_j u_{n+j} && \text{(line 18, } A\text{–orthogonalise against } q_1, \ldots, q_{k-1}\text{)} \\
\mu_{i,k} &= \psi_i - \sum_{j=1}^{k-1} \alpha_j \mu_{i,j}^c, \quad i = k, \ldots, s, && \text{(line 20, new column } k \text{ of } M \text{ using } M_c\text{)}
\end{aligned}$$

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

$$r_{n+k+1} = r_{n+k} - \frac{\phi_k}{\mu_{k,k}} g_{n+k}, \qquad \text{(line 23)} \tag{27}$$

since

$$\begin{aligned}
q_k^H r_{n+k+1} &= q_k^H r_{n+k} - \frac{\phi_k}{\mu_{k,k}} q_k^H g_{n+k} && (28) \\
&= q_k^H r_{n+k} - q_k^H r_{n+k} = 0 && (29)
\end{aligned}$$

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

$$x_{n+k+1} = x_{n+k} + \frac{\phi_k}{\mu_{k,k}} u_{n+k} \qquad \text{(line 24)} \tag{30}$$

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

$$\phi_i \leftarrow \phi_i - \frac{\phi_k}{\mu_{k,k}} \mu_{i,k}, \qquad i = k+1, \ldots, s \qquad \text{(line 28)} \tag{31}$$

which concludes the generation of the s vectors for $\mathcal{G}_{j+1}$. Therefore, using only the inner products ψ_i, every other inner product can be computed using only scalar updates.

As in Alg. 1, the computation of the (now solely combined) inner products is highlighted by boxes in Alg. 2. The small system in line 10 and in line 16 involving (part of) M 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 the remaining elements are used. This shows that there is now a *single* global synchronisation point per iteration step. The number of operations is the same as the original IDR(s)-biortho method in Alg. 1.

2.2 Finding the optimal parameter s

In this section a performance model is derived to compute the total execution time of the IDR(s)-biortho variant and the IDR(s)-minsync variant shown in Alg. 1 and Alg. 2, respectively. The performance model is used to estimate the optimal parameter s.

The model is based on message passing between single core computational nodes. The cubical domain is partitioned into rectangular cuboids and each domain is assigned to a single core. 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

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

where T_{comm} denotes the total communication time of the algorithm.

In parallel iterative methods there are two operations that require communication, 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 problem. Note that the model does not include preconditioning.

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. For the inner product it is assumed that each processor broadcasts its partial inner product to all the other inner products. Hence the following communication time for an inner product is obtained

$$T_{\text{dot}} = (p-1)(l + 8/b). \tag{34}$$

In the algorithms, some inner products can be combined. The time for the combined broadcasting of c partial inner products to p processors can be written as

$$T_{\text{cdot}} = (p-1)(l + 8c/b). \tag{35}$$

Assuming that a minimal residual strategy is used for computing ω, the total communication time spent on inner products in each cycle of IDR(s)-biortho is then

$$\begin{aligned} T_{\text{dots}}^{\text{biortho}}(p,s) &= (p-1)[(\tfrac{1}{2}s(s+1)+2)l + 8(s^2+s+2)/b] && (36)\\ &= (p-1)[\mathcal{O}(s^2)l + \mathcal{O}(s^2)/b]. && (37) \end{aligned}$$

For the IDR(s)-minsync variant it is

$$\begin{aligned} T_{\text{dots}}^{\text{minsync}}(p,s) &= (p-1)\left[(s+1)l + 8(s^2+s+2)/b\right] && (38)\\ &= (p-1)[\mathcal{O}(s)l + \mathcal{O}(s^2)/b]. && (39) \end{aligned}$$

The expressions (37) and (39) 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, while the time per cycle behaves quadratically in s for IDR(s)-biortho.

Since the domain is partitioned using a block partitioning, the time for the matrix–vector multiplication for an interior subdomain is (in both variants)

$$T_{\text{mmult}}(p) = 6\left(l + \frac{8n_x^2}{b\sqrt[3]{p}}\right), \tag{40}$$

if $n_x = n_y = n_z$.

To complete the performance model, the total number of dimension reduction steps has to be obtained. 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 N/s dimension reduction steps [14, §3]. In practical applications, the iteration process will exhibit much faster (superlinear) convergence rates according to $\widehat{N}/s$, where $\widehat{N} \ll N$. The total theoretical computing time on p processors for a given value of s is then

$$T_{\text{total}}(p,s) = \frac{\widehat{N}}{s} \times \left[\frac{\widetilde{T}(1,s)}{p} + T_{\text{dots}}(p,s) + (s+1)T_{\text{mmult}}(p)\right], \tag{41}$$

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

Using the expression (41), 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 merely a constant. Therefore, it does not play any role in minimising (41). Only problem and machine–dependent parameters are needed to find the optimal parameters s and p.

In multi–cluster environments, *intercluster* latencies are often several orders of magnitude higher than *intracluster* latencies. This is also true for the DAS–3 and the performance model is adapted accordingly. When estimating total computing times on a *single* cluster, the values for latency and bandwidth of this cluster are used in the performance model.

Additionally, when more clusters are added in the experiments, intracluster latency is neglected in the performance model. A cluster is considered a single 'supernode' and it is assumed that each cluster consists of a fixed amount P of 'regular' nodes. This means that in the model each supernode has the combined computational speed of P nodes. The intercluster latency and intercluster bandwidth of the DAS–3 is then used in the model to compute the optimal s and corresponding *number of clusters*.

2.3 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 $v = Q^H r$ for some vector r. The $N \times s$ matrix Q can be chosen arbitrarily and this freedom can be exploited [14, §4.1]. By using sparse column vectors for Q, the total cost of the inner products may be reduced significantly in the context of Grid computing. 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 $v = Q^H r$ 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.

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:

$$Q^H r = \left[\begin{array}{cc|cc|cc} q_1^a & q_1^b & & & & \\ q_2^a & q_2^b & & & & \\ \hline & & q_3^c & q_3^d & & \\ & & q_4^c & q_4^d & & \\ \hline & & & & q_5^e & q_5^f \\ & & & & q_6^e & q_6^f \end{array}\right] \times \left[\begin{array}{c} r^a \\ - \\ r^b \\ \hline r^c \\ - \\ r^d \\ \hline r^e \\ - \\ r^f \end{array}\right]. \tag{42}$$

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 the cluster. These results are then exchanged between the three clusters and broadcasted locally to the two nodes in each cluster.

3 Numerical experiments

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

(i) the IDR(s)-biortho variant given in Alg. 1 and using a dense matrix Q;

(ii) the IDR(s)-minsync variant given in Alg. 2 and using a dense matrix Q;

(iii) the IDR(s)-minsync variant given in Alg. 2 and using the sparse matrix Q as described in Sect. 2.3.

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

This section is structured as follows. In Sect. 3.1 and Sect. 3.2 a description is given of the target hardware and test problem, respectively. In Sect. 3.3 the parameter $\widehat{N}$ for this test problem and the (true) computational speed of a single core are estimated. In Sect. 3.4 the optimal parameter s for a particular computational environment is computed using the performance model. In Sect. 3.5 the performance model is compared to the numerical results. In Sect. 3.6 the experimental results are investigated more closely by comparing the time per cycle to the performance model. Finally, both strong and weak speedup results are given in Sect. 3.7, which includes results for sparse Q.

3.1 Target hardware

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

Table 1: Specific details on each DAS–3 site [10].

	LU site
1–way latency MPI (μsec)	2.7
max. throughput (MB/sec)	950
Gflops (HPL benchmark [9])	6.9
	DAS–3
WAN bandwidth (Mb/s)	40000
WAN latencies (μsec) (average)	990

Table 2: Specifications DAS–3: values (except WAN latencies) courtesy of Henri Bal [1].

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 [10]. 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 Tab. 1.

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 may contain multiple cores, we always employ a single core on each node for our computations. Table 2 lists values provided by Henri Bal on latency and bandwidth for the LU site and for the wide–area bandwidth on all five clusters [1]. These values were corroborated by the authors using the Intel MPI Benchmarks suite (IMB v2.3). The value for the WAN latency in Tab. 2 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 [17].

The algorithms are implemented using OpenMPI v1.2.1 and the implementations are matrix–free. Level 3 optimisation is used by the underlying GNU C compiler.

3.2 Test problem and experimental setup

# nodes	$p_x \times p_y \times p_z$	# equations
1	$1 \times 1 \times 1$	
2	$2 \times 1 \times 1$	
4	$2 \times 2 \times 1$	
8	$2 \times 2 \times 2$	
16	$4 \times 2 \times 2$	128^3
32	$4 \times 4 \times 2$	
64	$4 \times 4 \times 4$	
96	$4 \times 6 \times 4$	
128	$4 \times 8 \times 4$	

Table 3: Processor grid and problem size for the strong scalability experiments.

# nodes	$p_x \times p_y \times p_z$	# equations
30	$5 \times 3 \times 2$	398^3
60	$5 \times 4 \times 3$	501^3
90	$5 \times 6 \times 3$	574^3
120	$5 \times 6 \times 4$	631^3
150	$5 \times 6 \times 5$	680^3

Table 4: Processor grid and problem sizes for the weak scalability experiments.

Consider the following three–dimensional elliptic partial differential equation taken from [13]:

$$\nabla^2 u + w u_x = f(x, y, z), \tag{43}$$

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

$$u = \exp(xyz) \sin(\pi x) \sin(\pi y) \sin(\pi z) \tag{44}$$

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

Discretisation 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 orthogonalised random vectors. The iteration is terminated when $||r_n||/||r_0|| \leq \varepsilon \equiv 10^{-6}$ 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.

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.

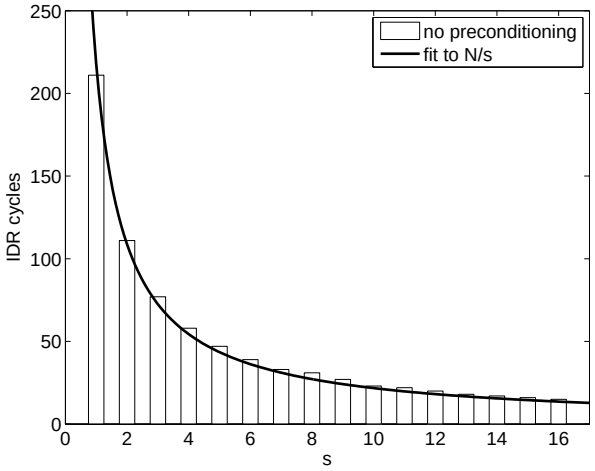
The experiments for investigating strong scalability are performed using the four sites that employ the fast interconnect. On each site, 32 nodes are used, which gives a total of 128 nodes for the largest experiment. Experiments that use less than 32 nodes are performed on the LU site and in each subsequent 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 sometimes included, which means that in this case the slower interconnect is used. The number of equations per node is set to approximately two million equations, yielding the problem sizes shown in Tab. 4.

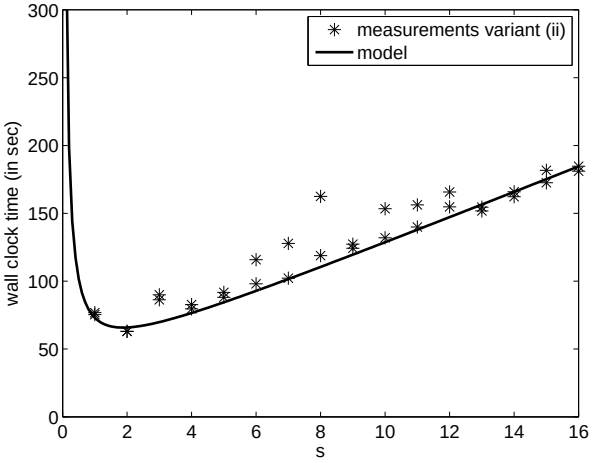
The computational domain is partitioned using a three–dimensional block partitioning, where the nodes are arranged in a non–periodic Cartesian grid $p_x \times p_y \times p_z$. Table 3 lists the dimensions of the processor grid for each number of nodes used in the strong scalability experiments. The domain is partitioned in such a way that the partition size along the x direction is always constant on each site. Similarly, Tab. 4 lists the sizes of the processor grids for the weak scalability experiments.

3.3 Estimating parameters of the 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 Fig. 1(a). These experimental data are obtained using



(a) Total IDR cycles, fitted to $\widehat{N}/s$ with $\widehat{N} \approx 218$.



(b) Modeling computing times single node.

Figure 1: Estimating model parameters: $\widehat{N}$ and processor speed.

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 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 Fig. 1(b). Since variants (i) and (ii) have the same number of operations, only results from the second variant are given. Also shown in Fig. 1(b) are the theoretical computing times of the algorithm, which is equal to

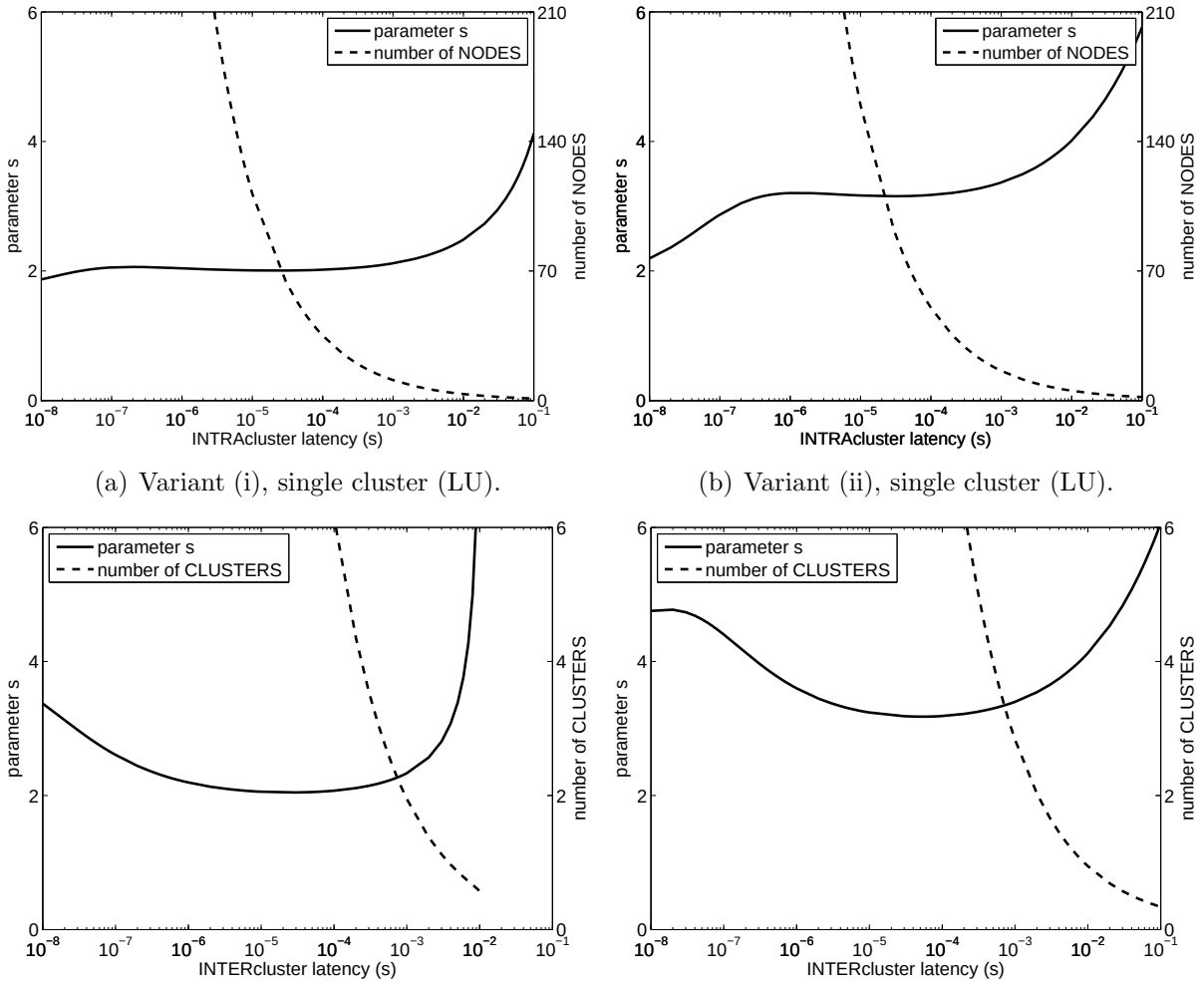
$$\frac{\widehat{N}}{s} \times \widetilde{T}(1, s), \tag{45}$$

using the previously obtained value of $\widehat{N}$.

According to the HPL benchmark (see Tab. 2), the peak performance of a single core is 6.9 Gflops. In realistic applications, it is not uncommon that only a fraction of the peak performance can be attained. When using the more realistic processor performance of 3×10^{-1} Gflops, the model (solid line) corresponds to the experimental data perfectly. The results indicate that when using a single core, setting $s = 2$ results in the fastest computing times.

3.4 Estimating the optimal parameter s

By using the expression for the theoretical computing time (41) from Sect. 2.2, the calibrated value for the computational speed from Sect. 3.3, and data from Tab. 2 for latency



(a) Variant (i), single cluster (LU).

(b) Variant (ii), single cluster (LU).

(c) Variant (i), multi–cluster (DAS–3).

(d) Variant (ii), multi–cluster (DAS–3).

Figure 2: A–priori estimation of optimal parameter s and corresponding number of nodes/clusters.

and bandwidth, the optimal parameter s that minimises 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 Sect. 2.2, estimates for both a single cluster (the LU site) and for a multi–cluster (the DAS–3) will be given. Using data from Tab. 2, the optimal s and corresponding (theoretical) number of nodes for the LU cluster are computed and shown in Fig. 2(a) and 2(b) for variants (i) and (ii), respectively. Similarly, Fig. 2(c) and 2(d) show the optimal s and corresponding (theoretical) number of clusters for the DAS–3 multi–cluster — with 32 nodes per cluster — for variants (i) and (ii), respectively.

Note that according to the model results from Fig. 2(a) and 2(b), the optimal s for

extremely small latencies on a single cluster approaches $s = 2$ for both variants. This result is in agreement with Fig. 1(b) from Sect. 3.3, where the computing times on a *single* node (i.e., latency equal to *zero*) were given. Here, the optimal values of s was also $s = 2$.

Not surprisingly, the results show that when latency is low, it is beneficial to use a large value of s and a small number of nodes on a single cluster. The same holds for the DAS–3 multi–cluster. A reason is that in both these cases, the overhead when generating the vectors for $\mathcal{G}_j$ is low. This phenomenon can be observed for both variant (i) and (ii).

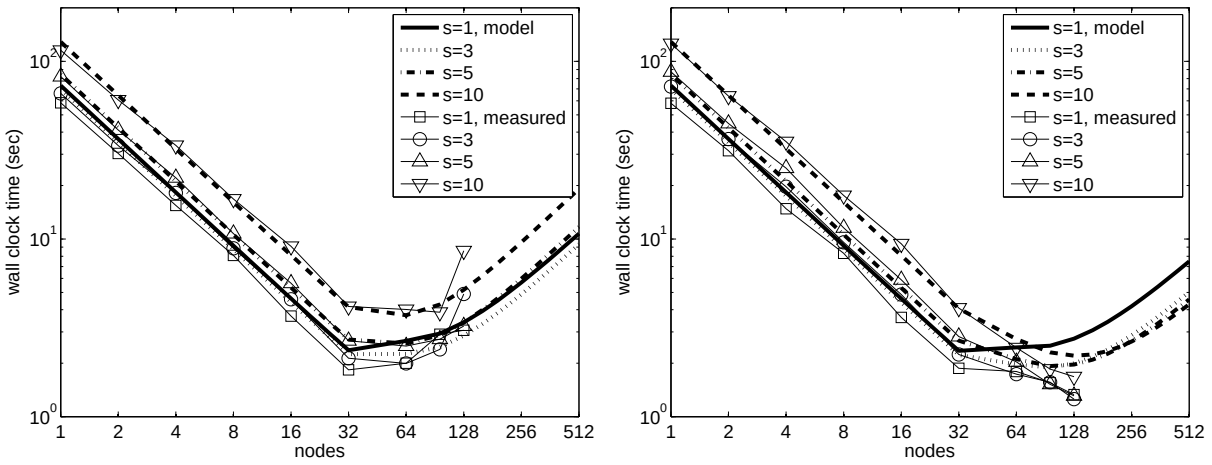
Note that for latencies $l > 10^{-4}$ and using variant (ii), the model for the multi–cluster gives roughly the same optimal s as the single cluster model.

In multi–cluster environments, intercluster latency is often relatively high. Figure 2(c) shows that in this case variant (i) is completely ineffective, since the value of s approaches infinity. For variant (ii) behaves much more favourably for low latencies, where the value of s approaches $s = 6$.

Although it is not particularly natural to give results for extremely low latencies in the multi–cluster model, they are included for completeness.

It is interesting to note that in parallel IDR(s) methods on cluster and Grid computing the optimal value of s and the corresponding number of nodes/clusters can be determined in such a manner.

3.5 Validation of the parallel performance model for Grid computing



(a) Variant (i).

(b) Variant (ii).

Figure 3: Performance model results for variants (i) and (ii).

Shown in Fig. 3 are the predicted total computing times defined by (41) from Sect. 2.2 for variants (i) and (ii) using $s \in \{1, 3, 5, 10\}$ and using up to (in theory) 512 nodes (i.e., sixteen clusters). As mentioned before, experiments that use up to 32 nodes employ

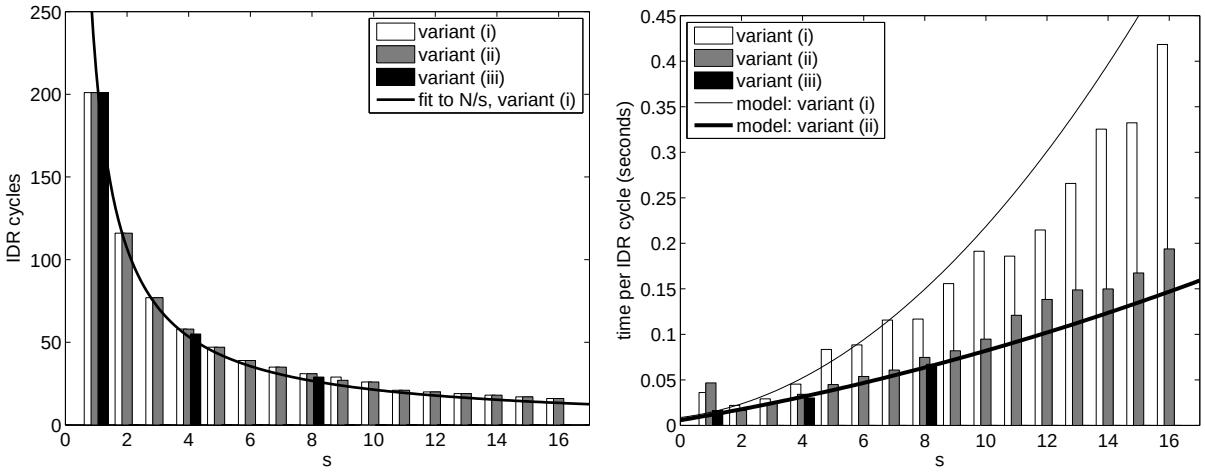
the LU site and corresponding parameters in the performance model. In each additional experiment, a cluster is added consisting of 32 nodes and the corresponding multi–cluster performance model is used. Also shown are the measured wall clock times using up to 128 nodes (i.e., four DAS–3 clusters).

Communication overhead on the LU site (i.e., $p \leq 32$) 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. However, when more clusters are added (i.e., $p > 32$), the effect of communication begins to play a more significant role. Clearly, the optimal s and number of nodes differ for both variants.

According to the measurements from Fig. 3(a), the optimal optimal value of s for solving the 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 Fig. 2(c), which shows that for the DAS–3 latency $l = 990 \times 10^{-6}$s given in Tab. 2, the optimal value of s is close to $s = 2$ using two clusters.

Similarly, measurements from Fig. 3(b) show that for variant (ii) the optimal value of s is between $s = 3$ and $s = 5$. Correspondingly, the optimal number of nodes is between 96 and 128 nodes (i.e., between three and four clusters). The predictions in Fig. 2(d) show that for $l = 990 \times 10^{-6}$s the optimal value of s lies close to $s = 4$ using three to four clusters. This also corresponds well to the measurements.

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



(a) Total IDR cycles, $\widehat{N} \approx 211$.

(b) Time per IDR cycle.

Figure 4: Investigating s–dependence for $s \in \{1, \ldots, 16\}$ using 64 nodes, four sites, and fast network.

To investigate the relation between the value of s and the time per IDR(s) cycle, the following experiment is performed. Figure 4 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.

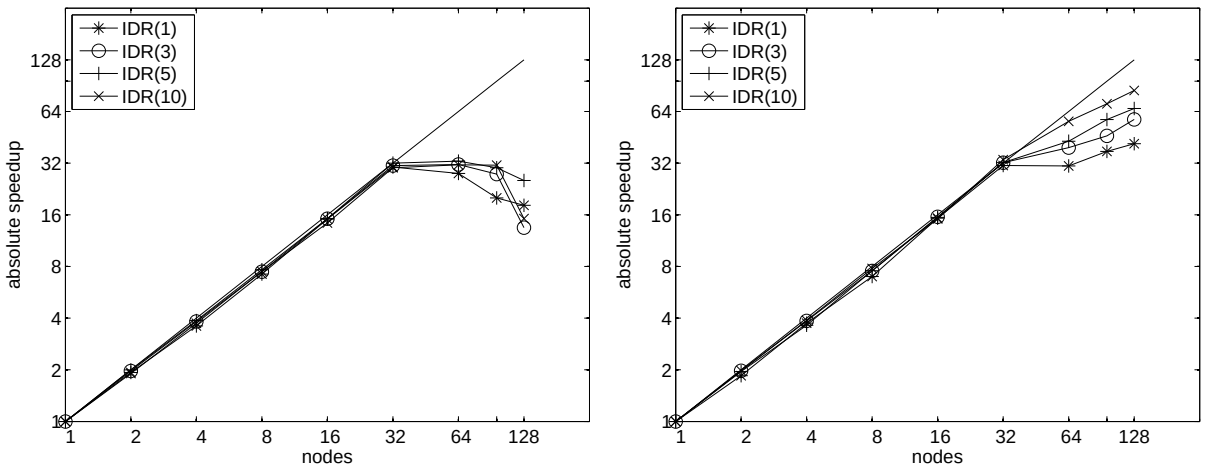
For completeness, the total number of IDR cycles is shown in Fig. 4(a), which is practically identical for all three variants. The experiments showed that the use of sparse column vectors for Q can result in numerical instability issues for large s. That is, when using variant (iii) the iteration did 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 Sect. 3.3.

More interestingly, Fig. 4(b) shows the wall clock time per IDR cycle for increasing values of s. As mentioned in Sect 2.2, the performance model (i.e., expression (39)) 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 (37) from Sect 2.2 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 Fig. 4(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 optimisations, since *disabling* these optimisations *reduced* the time per cycle. For some reason, variant (iii) conforms well to the performance model for $s = 1$ when using the compiler optimisations.

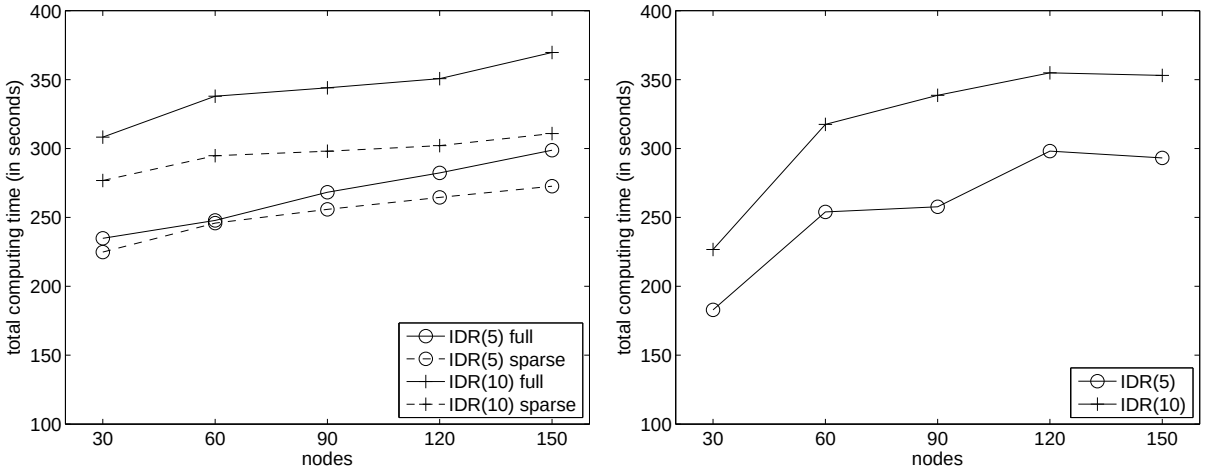
3.7 Parallel speedup results



(a) Variant (i).

(b) Variant (ii).

Figure 5: Absolute speedup, scaled to number of iterations and using a log–log scale.



(a) Comparing variant (ii) and (iii), first strategy.

(b) Variant (ii), second strategy.

Figure 6: Weak scaling experiments on DAS–3 for variants (ii) and (iii).

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

$$S(p) = \frac{T(1)}{T(p)}, \tag{46}$$

where $T(1)$ is the execution time of the parallel algorithm on one node and p is the number of nodes. Figure 5 shows strong scalability results using variants (i) and (ii) for $s \in \{1, 3, 5, 10\}$, in Fig. 5(a) and Fig. 5(b) respectively. The scalability results of variant (iii) is 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 variant (ii) scales much more favourably than 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.

To investigate weak scalability, 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 Tab. 4. Note that for variant (iii), the value of s has to be a multiple of the number of clusters in the grid.

The problem size is increased using two different strategies. The first strategy equally divides the nodes between the five DAS–3 sites, starting with six nodes per site for the smallest experiment. This allows comparing the use of a dense matrix Q and of a sparse matrix Q. In the second strategy, one whole site is added each time, starting with the LU

site as before and finishing with the TUD site. In the ideal case, the time per iteration is constant for increasing number of nodes.

Figure 6 shows the weak speedup results for $s \in \{5, 10\}$. In Fig. 6(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. In addition, the time per iteration becomes increasingly constant for larger values of s, showing favourable weak scalability.

Figure 6(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 servers show weak scalability results comparable to Fig. 6(a).

4 Conclusions

The recent IDR(s) method is a family of fast algorithms for solving large sparse nonsymmetric linear systems. On cluster and in particular on Grid computers, global synchronisation is a critical bottleneck in parallel iterative methods. To alleviate this bottleneck in IDR(s) algorithms, three strategies were used. Firstly, by reformulating the efficient and numerically stable IDR(s)-biortho method [16], the IDR(s)-minsync method was derived which has a *single* synchronisation 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 IDR(s)-biortho variant has no speedup whatsoever on the DAS–3 multi–cluster.

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 minimised on the DAS–3 multi–cluster. The experiments show that this approach results in reduced execution times, in particular for larger values of s.

Using only problem and machine–dependent parameters, the presented parallel performance model can be utilised for a–priori esimation of the optimal value of s and number of nodes. In this way, the total execution time on the DAS–3 multi–cluster of both IDR(s) variants can be minimised.

Acknowledgments

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 Organisation for Scientific Research (NWO/NCF) is gratefully acknowledged for the use of the DAS–3 system.

References

[1] Henri Bal. DAS–3 opening symposium, 2007. `http://www.cs.vu.nl/das3/symposium07/das3-bal.pdf`. Retrieved 05/02/2009.

[2] A. T. Chronopoulos and C. W. Gear. *S*–step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.*, 25(2):153–168, 1989.

[3] E. de Sturler. A performance model for krylov subspace methods on mesh-based parallel computers. *Parallel Comput.*, 22(1):57–74, 1996.

[4] Tong-Xiang Gu, Xian-Yu Zuo, Xing-Ping Liu, and Pei-Lu Li. An improved parallel hybrid bi–conjugate gradient method suitable for distributed parallel computing. *J. Comput. Appl. Math.*, 226(1):55–65, 2009.

[5] Tong-Xiang Gu, Xian-Yu Zuo, Li-Tao Zhang, Wan-Qin Zhang, and Zhiqiang Sheng. An improved bi–conjugate residual algorithm suitable for distributed parallel computing. *Applied Mathematics and Computation*, 186(2):1243–1253, 2007.

[6] Martin H. Gutknecht. IDR Explained. *Electr. Trans. Numer. Anal.*, October 2009. (to appear).

[7] Yusuke Onoue, Seiji Fujino, and Norimasa Nakashima. Improved IDR(s) method for gaining very accurate solutions. *World Academy of Science, Engineering and Technology*, 55, 2009.

[8] Yusuke Onoue, Seiji Fujino, and Norimasa Nakashima. An overview of a family of new iterative methods based on IDR theorem and its estimation. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol II IMECS 2009, March 18 - 20, 2009, Hong Kong*, 2009.

[9] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL – a portable implementation of the high–performance Linpack benchmark for distributed–memory computers, 2008. Available at `http://www.netlib.org/benchmark/hpl/`.

[10] Frank J. Seinstra and Kees Verstoep. DAS–3: The distributed ASCI supercomputer 3, 2007. `http://www.cs.vu.nl/das3/`.

[11] Valeria Simoncini and Daniel B. Szyld. Interpreting IDR as a Petrov–Galerkin method. Technical Report 09-10-22, Department of Mathematics, Temple University, October 2009.

[12] Gerard L. G. Sleijpen, Peter Sonneveld, and Martin B. van Gijzen. Bi–CGSTAB as an induced dimension reduction method. *Applied Numerical Mathematics*, In Press, Corrected Proof:–, 2009.

[13] G.L.G. Sleijpen and D.R. Fokkema. BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 1:11–32, 1993.

[14] Peter Sonneveld and Martin B. van Gijzen. IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 31(2):1035–1062, 2008.

[15] H. A. van der Vorst. 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*, 13(2):631–644, 1992.

[16] Martin B. van Gijzen and Peter Sonneveld. An elegant IDR(s) variant that efficiently exploits bi–orthogonality properties. Technical report, Delft University of Technology, Delft, the Netherlands, 2008. DUT report 08–21.

[17] Kees Verstoep, Jason Maassen, Henri E. Bal, and John W. Romein. Experiences with fine–grained distributed supercomputing on a 10G testbed. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 376–383, Washington, DC, USA, 2008. IEEE Computer Society.

[18] L. Yang and R. Brent. The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. In *5th International Conference on Algorithms and Architectures for Parallel Processing*, pages 324–328, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[19] T. R. Yang. The improved CGS method for large and sparse linear systems on bulk synchronous parallel architecture. In *ICA3PP '02: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*, pages 232–237, Washington, DC, USA, 2002. IEEE Computer Society.

[20] Tianruo Yang and Hai-Xiang Lin. The improved quasi-minimal residual method on massively distributed memory computers. In *HPCN Europe '97: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, pages 389–399, London, UK, 1997. Springer-Verlag.