WEIGHTED BLOCK-ASYNCHRONOUS RELAXATION FOR GPU-ACCELERATED SYSTEMS[*]

HARTWIG ANZT[†], STANIMIRE TOMOV[‡], JACK DONGARRA[‡§], AND VINCENT HEUVELINE[†]

Abstract. In this paper, we analyze the potential of using weights for block-asynchronous relaxation methods on GPUs. For this purpose, we introduce different weighting techniques similar to those applied in block-smoothers for multigrid methods. Having proven a sufficient convergence condition for the weighted block-asynchronous iteration, we analyze the performance of the algorithms implemented using CUDA and compare them with weighted synchronous relaxation schemes like SOR. For test matrices taken from the University of Florida Matrix Collection we report the convergence behavior and the total runtime for the different weighting techniques. Analyzing the results, we observe that using weights may accelerate the convergence rate of block-asynchronous iteration considerably. This shows the high potential of using weights in block-asynchronous iteration for numerically solving linear systems of equations fulfilling certain convergence conditions. While component-wise relaxation methods are seldom directly applied to linear equation systems, using them as smoother in a multigrid framework they often provide an important contribution to finite element solvers. Since the parallelization potential of the classical smoothers like SOR and Gauss-Seidel is usually very limited, replacing them with block-asynchronous smoothers may have a considerable impact on the overall multigrid performance. Due to the explosion of parallelism in today's architecture designs, the significance and the need for highly parallel asynchronous smoothers, as the ones described in this work, is expected to grow.

Key words. asynchronous relaxation, weighted block-asynchronous iteration methods, multigrid smoother, GPU

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction.

Using weights in iterative relaxation schemes is a well known and often applied technique to improve the convergence. An example is the classical successive over-relaxation method (SOR, [26]), which improves the convergence rate of the underlying Gauss-Seidel algorithm by using weights. Other examples include block-smoothers in multigrid for finite element discretizations [9]. In this case, the parallelized Block-Jacobi or Block-Gauss-Seidel smoothers are weighted according to the block decomposition of the the linear equation system.
In [3] we explored the potential of replacing the classically applied smoothers in multigrid methods with asynchronous iterations. While a block parallelized smoother requires synchronization between the iterations, asynchronous methods are very tolerant to component update order and latencies concerning the communication of updated component values. This lack of synchronization barriers makes them perfect candidates for modern hardware architectures, which are often accelerated by highly parallel coprocessors. In [2] we have shown how to enhance asynchronous iteration schemes to compensate for the inferior convergence rate of the plain asynchronous iteration.

[*]The Numerical Experiment Results in this paper are partly used in a paper submitted to the International Conference on Computational Science (ICCS 2012) and the Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA 2012).

[†]Karlsruhe Institute of Technology (KIT), Institute for Applied and Numerical Mathematics 4, Fritz-Erler-Str. 23, 76133 Karlsruhe, Germany.

[‡]University of Tennessee, Innovative Computing Lab, 1122 Volunteer Blvd., Knoxville TN 37996-3450, USA.

[§]Oak Ridge National Laboratory, Oak Ridge TN 37831, USA. University of Manchester, Oxford Road, Manchester M13 9PL, UK.

In particular, this is achieved by adding local iterations that almost come for free, and therefore should not be counted as global iterations. Furthermore, the higher iteration number per time frame on the GPUs results in considerable performance increase.

While Chazan and Miranker have introduced a weighted asynchronous iteration similar to SOR, it becomes of interest as whether or not the block-asynchronous iteration benefits from weighting methods similar to those applied to block smoothers [9]. The motivation is that the off-block matrix entries are neglected in the local iterations performed on the subdomains. To account for this issue it may be beneficial to weight the local iterations. This can be achieved either by using ℓ_1-weights, by a technique similar to ω-weighting, or by a combination of both. The purpose of this paper is to introduce the different methods, analyze the convergence properties, and report experimental results.
Therefore, we split the paper into different parts. In the first section we provide the concept of asynchronous iteration and state some established convergence results on the weighted asynchronous iteration method [12]. We then introduce the idea of block-asynchronous methods based on the linear system decomposition and the different weighting techniques. Having given the necessary mathematical background, we will then prove the convergence of the block-asynchronous iteration using ω or ℓ_1 weights. In the experimental part we provide details about the hardware system, the implementation we used for our tests and describe the linear equation systems we target. We then report the convergence rate with respect to the iteration number for different parameters, and compare the time to solution with a SOR method implemented on the CPU.

2. Mathematical Background.

2.1. Asynchronous Iteration. The Jacobi method is an iterative algorithm for finding the approximate solution for a linear system of equations

$$Ax = c, \tag{2.1}$$

that converges if A is strictly or irreducibly diagonally dominant [30, 6].

One can rewrite the system as $(L + D + U)x = c$ where D denotes the diagonal entries of A while L and U denote the lower and upper triangular part of A, respectively. Using the form $Dx = c-(L+U)x$, the Jacobi method is derived as an iterative scheme

$$x^{m+1} = D^{-1}(c - (L + U)x^m).$$

Denoting the error at iteration $m + 1$ by $e_{m+1} \equiv x^{m+1} - x$, this scheme can also be rewritten as $e_{m+1} = (I - D^{-1}A)e_m$. The matrix $M \equiv I - D^{-1}A$ is often referred to as *iteration matrix*. The Jacobi method provides a sequence of solution approximations with increasing accuracy when the spectral radius of the iteration matrix M is less than one (i.e., $\rho(M) < 1$) [6].

The Jacobi method can also be rewritten in the following component-wise form:

$$\begin{aligned} x_i^{m+1} &= \frac{1}{a_{ii}} \left(c_i - \sum_{j \neq i} a_{ij} x_j^m \right) && (2.2) \\ &= \sum_{j \neq i} b_{ij} x_j^m + d_i && (2.3) \end{aligned}$$

where $b_{ij} = -\frac{a_{ij}}{a_{ii}}$ for $i \neq j$, $b_{ii} = 0$ and $d_i = \frac{c_i}{a_{ii}}$ for all $1 \leq i, j \leq n$.

For computing the next iteration, one requires the latest values of all components. This requires a strict order of the component updates, limiting the parallelization potential to a stage, where no component can be updated several times before all the other components are updated.

If this order is not adhered to, i.e the individual components are updated independently and without consideration of the current state of the other components, the resulting algorithm is called chaotic or asynchronous iteration method. Back in the 70's Chazan and Miranker analyzed some basic properties of these methods, and established convergence theory [12] (also see [4, 10, 11]). For the last 30 years, these algorithms came out of focus of high-performance computing due to the superior convergence properties of synchronized iteration methods. More interest was put on the convergence properties and deriving models for the computational cost, e.g. in [5, 7, 13, 14, 15, 16, 17, 18, 19, 21, 22, 28]. Today, due to the complexity of heterogeneous hardware platforms and the high number of computing units in parallel devices like GPUs, these schemes may become interesting again for applications like multigrid methods, where highly parallel smoothers are required on the distinct grid levels. While traditional smoothers, like the sequential Gauss-Seidel, obtain their efficiency from their fast convergence, it may be true that the asynchronous iteration scheme overcompensates for the inferior convergence behavior with superior scalability.

The chaotic-, or asynchronous-relaxation scheme defined by Chazan and Miranker [12] can be characterized by two functions, an update function $u(\cdot)$ and a shift function $s(\cdot,\cdot)$. For each non-negative integer ν, the component of the solution approximation x that is updated at step ν is given by $u(\nu)$. For the update at step ν, the m^{th} component used in this step is $s(\nu, m)$ steps back. All the other components are kept. This can be expressed as:

$$x_l^{\nu+1} = \begin{cases} \sum_{m=1}^{n} b_{l,m} x_m^{\nu - s(\nu,m)} + d_l & \text{if } l = u(\nu) \\ x_l^{\nu} & \text{if } l \neq u(\nu). \end{cases} \tag{2.4}$$

Furthermore, the following conditions can be defined to guarantee the well-posedness of the algorithm [27]:

Definition 2.1 (Conditions for well-posedness of asynchronous iteration).

1. *The update function $u(\cdot)$ takes each of the values l for $1 \leq l \leq n$ infinitely often.*
2. *The shift function $s(\cdot,\cdot)$ is bounded by some $\bar{s}$ such that $0 \leq s(\nu, m) \leq \bar{s} \forall\ \nu \in \{1, 2, \dots\}, \forall m \in \{1, 2, \dots n\}$. For the initial step, we additionally require $s(\nu, m) \leq \nu$.*

For the general asynchronous iteration defined by these properties, Chazan and Miranker have proved the following sufficient convergence condition (See also Bertsekas and Tsitsiklis for the proof [11]).

Theorem 2.2. *Suppose we have an asynchronous iteration where condition (1) and (2) are satisfied for all update and shift functions.*

(a) The sequence of iterates x^t converges if there exists a positive vector v and a number α, $\alpha < 1$ such that $|B|v \leq \alpha v$ (componentwise).

(b) This happens if the spectral radius of $|B|$, $\rho(|B|) < 1$.

Without showing the details of the proof, we want to mention a lemma that is used, [12, 27]:

Lemma 2.3. *The condition $\rho(|B|) < 1$ implies that there is a value α with $0 < \alpha < 1$ and a vector v with positive components such that $|B|v \leq \alpha v$ (componentwise).*

Chazan and Miranker extended their results to a weighted asynchronous iteration by replacing the original iteration matrix $B = I - D^{-1}A$ of (2.4) by $B_\omega = I - \omega D^{-1}A$. They were also able to prove the convergence of the obtained weighted asynchronous iteration [12].

Theorem 2.4. *Suppose the asynchronous iteration fulfilling condition (1) and (2) is modified by using weights such that $B = I - D^{-1}A$ and $d = D^{-1}c$ of (2.4) is replaced by $B_\omega = (I - \omega D^{-1}A)$ and $d_\omega = \omega D^{-1}c$.*
If the spectral radius $\rho(|B|) = \alpha < 1$, then the weighted asynchronous iteration converges for all ω with $0 < \omega < \frac{2}{\alpha+1}$.

We outline the proof here, since we will utilize parts in the convergence theory of weighted block-asynchronous iteration.

Proof. We want to show that $\rho(|B_\omega|) < 1$ or alternatively that there exists $v > 0$ so that $|B_\omega|v < \beta v$, $\beta < 1$. For the case $\omega = 1$ we obtain from Theorem 2.2 that there exists $v > 0$ so that $|B_1|v < \alpha v$. But then

$$|B_\omega|v \leq (I(1-\omega) + \omega|B_1|)v \leq |(1-\omega)|v + \omega\alpha v = (|1-\omega| + \omega\alpha)v.$$

Let $\beta = (|1-\omega| + \omega\alpha)$. It remains to show that $\beta < 1$. If $1 < \omega < \frac{2}{\alpha+1}$, then $\beta = \alpha\omega + (\omega - 1) = (1+\alpha)\omega - 1 < 1$. On the other hand, if $0 \leq \omega \leq 1$ we have $\beta = \alpha\omega + (1-\omega) = (-(1-\alpha)\omega + 1) < 1$ since $\alpha < 1$. $\square$

2.2. Block-Asynchronous Iteration. The motivation for the block-asynchronous iteration comes from the hardware architecture. The idea is to split the linear system into blocks of rows, and then to assign the computations for each block to one thread block on the GPU. For these thread blocks, an asynchronous iteration method is used, while on each thread block, instead of one, multiple Jacobi-like iterations are performed. During these local iterations the x values used from outside the block are kept constant (equal to their values at the beginning of the global iteration). After the local iterations, the updated values are communicated. This approach is inspired by the well known hybrid relaxation schemes [8, 9], and was analyzed with respect to convergence properties in [20]. In other words, using domain-decomposition terminology, our blocks would correspond to subdomains and thus we iterate locally on every subdomain. We denote this scheme by *async-(i)*, where the index i indicates that we use i Jacobi-like updates on the subdomain. As the subdomains are relatively small and the data needed largely fits into the multiprocessor's cache, these additional iterations on the subdomains almost come for free. The obtained algorithm, visualized in Figure 2.1, can be written as component-wise update of the solution approximation:

$$x_k^{(m+1)} += \frac{1}{a_{kk}} \left(\underbrace{c_k - \sum_{j=1}^{T_S} a_{kj} x_j^{(m-\nu(m+1,j))}}_{\text{global part}} \underbrace{- \sum_{j=T_S}^{T_E} a_{kj} x_j^{(m)}}_{\text{local part}} \underbrace{- \sum_{j=T_E}^{n} a_{kj} x_j^{(m-\nu(m+1,j))}}_{\text{global part}} \right),$$

where T_S and T_E denote the starting and the ending indexes of the matrix/vector part in the thread block. Furthermore, for the local components, the most recent values are used, while for the global part, the values from the beginning of the iteration are used. The shift function $\nu(m+1, j)$ denotes the iteration shift for the component j - this can be positive or negative, depending on whether the respective other thread block has already conducted more or less iterations. Note that this gives a block Gauss-Seidel

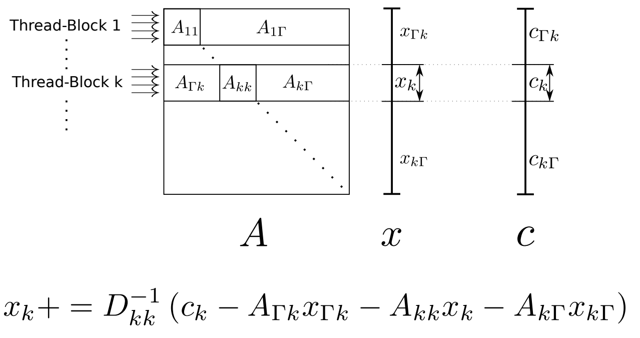


Fig. 2.1: Visualizing the asynchronous iteration in block description used for the GPU implementation.

flavor to the updates. It should also be mentioned that the shift function may not be the same in different thread blocks.

Despite the fact that classical relaxation methods are nowadays rarely applied to solve a linear equation system (there exist more efficient methods like Krylov subspace solvers) they are still of interest for high performance computing since they often provide an important contribution as a smoother in the multigrid framework. While experiment results reveal that block-asynchronous iteration may, especially when targeting large systems and highly parallel implementations, have similar smoothing potential like Gauss-Seidel [3], it is an open question of whether or not the performance of block-asynchronous iteration can be increased by using weighting techniques similar to those applied to hybrid Gauss-Seidel-based smoothers [9, 23].

To examine the topic of weights in the hybrid asynchronous iteration, we introduce the following notations to simplify the analysis [9].

Splitting the matrix A into blocks, we use A_{kk} for the matrix block located in the k-th block row and the k-th block column. We now define the sets

$$\begin{aligned}\Omega^{(i)} &= \{j \in \Omega_k : i \in \Omega_k\},\\ \Omega_0^{(i)} &= \{j \notin \Omega_k : i \in \Omega_k\}.\end{aligned}$$

Hence, for block A_{kk}, $\Omega^{(i)}$ contains all column indices in the diagonal block of row i while $\Omega_0^{(i)}$ contains the remaining columns that have no entries in the block. This way, we can decompose the sum of the elements of the i-th row:

$$\begin{aligned}\sum_j a_{ij} &= \underbrace{\sum_{j=1}^{T_S} a_{ij}}_{\text{off-block columns}} + \underbrace{\sum_{j=T_S}^{T_E} a_{ij}}_{\text{block columns}} + \underbrace{\sum_{j=T_E}^{n} a_{ij}}_{\text{off-block columns}}\\ &= \underbrace{\sum_{j\in\Omega^{(i)}} a_{ij}}_{\text{block columns}} + \underbrace{\sum_{j\in\Omega_0^{(i)}} a_{ij}}_{\text{off-block columns}} .\end{aligned}$$

Similar to [9] we may now define an indicator θ such that for all rows i

$$a_{ii} \geq \theta \sum_{j \in \Omega_0^{(i)}} |a_{ij}|$$

indicates a certain quality of the parallel partitioning of the matrix A. Large values for θ imply that most of the relatively significant entries in every row are within the respective diagonal blocks. For the hybrid asynchronous iteration using the matrix block decomposition this means that the off-thread entries that are neglected in the local iterations are relatively small.
For many cases $\theta > 1$, which we will also assume for further analysis, since it is a necessary condition for the convergence theory.

3. Weights in block-asynchronous iteration.

3.1. ω-weighting for block-asynchronous iteration. Similar to the ω-weighted asynchronous iteration, it is possible to use ω-weighting for the block structure in the hybrid approach. In this case, the solution approximation of the local iterations is weighted when updating the global iteration values. The parallel algorithm for the component updates in one matrix block is outlined in Algorithm 1.

```
1: Update component i:
2: $s := \sum_{j \in \Omega_0^{(i)}} b_{ij} x_j$ (off-diagonal part)
3: $x^{local} = x$
4: for all $k = 0;\ k < local\_iters;\ k++$ do
5:     $x_i^{local} := s + \sum_{j \in \Omega^{(i)}} b_{ij} x_j^{local}$ (using the local updates in the block)
6: end for
7: $x_i = \omega x_i^{local} + (1-\omega) x_i$
```

Algorithm 1: Basic principle of using ω-weights in block-asynchronous iteration.

For this algorithm it is difficult to derive one general iteration matrix, since for the local iterations $B = I - D^{-1}A$ is applied, and for the global updates $B_\omega = I - \omega D^{-1}A$ is used. Hence we obtain for one global iteration entailing ν local iterations

$$x^{t+1} = (B_\omega B^\nu) x^t.$$

THEOREM 3.1. *Suppose the block-asynchronous iteration fulfilling condition (1) and (2) for all update and shift functions is modified by using weights in the block approach such that $B = I - D^{-1}A$ and $d = D^{-1}c$ of (2.4) is replaced by $B_{h\omega} = (I - \omega D^{-1}M)(I - D^{-1}M)^\nu$ and $d_{h\omega} = \omega D^{-1} D^{-\nu} c$, respectively, where ν is the number of local iterations on every subdomain.*
If the spectral radius of $\rho(|B|) = \alpha < 1$, then the weighted block-asynchronous iteration converges for all ω with $0 < \omega < \frac{2}{\alpha+1}$.

Proof. Due to the general Theorem 2.2, it is sufficient to show that there exists $v > 0$ so that $|B_{h\omega}|v < \delta v$, $\beta < 1$.
Since we have that $\rho(|B|) = \alpha < 1$, we know according to Lemma 2.3 that there exists an $\alpha < 1$, and a $v > 0$ such that $|B|v < \alpha v$. We utilize this α and v and the β from

the proof of Theorem 2.4 fulfilling $|B_\omega|v < \beta v$. We then conclude for $|B_{h\omega}|v$:

$$\begin{aligned} |B_{h\omega}|v &= |B_\omega B^\nu|v \leq |B_\omega||B^\nu|v \\ &\leq |B_\omega||B|^\nu v < |B_\omega||B|^{\nu-1}\alpha v \\ &< |B_\omega|\alpha^\nu v < \beta\alpha^\nu v \end{aligned}$$

Let $\delta = \beta\alpha^\nu$, then $\delta < 1$ and we have the existence of $v > 0$ such that $|B_{h\omega}|v < \delta v$, completing the proof. □

3.2. ℓ_1-weighting for block-asynchronous iteration. Using the notation for the local and the global parts of the matrix, we can introduce a weighting technique, that is usually referred to as ℓ_1 weighting. Classically applied to Block-Jacobi and Gauss-Seidel relaxation methods, ℓ_1 weighting adds a diagonal component to the iteration matrix. This way, $B = I - D^{-1}A$ of (2.4) is replaced by $B = I - (D + D^{\ell_1})^{-1}A$, where D^{ℓ_1} is the diagonal matrix with entries

$$d_{ii}^{\ell_1} = \sum_{j \in \Omega_0^{(i)}} |a_{ij}|.$$

In other words, the iteration matrix M of $B = I - M^{-1}A$ is perturbed by adding the off-block elements in each row to the diagonal.

A question in this context is whether the obtained weighted asynchronous iteration is still convergent.

Theorem 3.2. *Suppose*

$$x_i^{\nu+1} = \sum_{j=1}^{n} b_{ij}^{\ell_1} x_j^{\nu - s(\nu,m)} + d_i^{\ell}, \quad i = 1, 2 \ldots n$$

is an ℓ_1 weighted asynchronous iteration scheme where condition (1) and (2) are fulfilled for all update and shift functions. If furthermore

$$\rho(|B|) < 1$$

where $\rho(|B|)$ is the spectral radius of the component-wise positive matrix $B = I - D^{-1}A$, then the sequence of solution approximations x_i^t is convergent to $\bar{x}_i$, the unique solution of 2.1.

The proof to this is very similar to the general theorem in [12] with the difference, that the iteration matrix B of (2.4) is replaced by

$$B_{\ell_1} = \begin{pmatrix} 1 - \frac{a_{11}}{a_{11}+d_{11}} & -\frac{a_{12}}{a_{11}+d_{11}} & -\frac{a_{13}}{a_{11}+d_{11}} & \cdots & -\frac{a_{1n}}{a_{11}+d_{11}} \\ -\frac{a_{21}}{a_{22}+d_{22}} & 1 - \frac{a_{22}}{a_{22}+d_{22}} & \ddots & \vdots & \\ -\frac{a_{31}}{a_{33}} & \ddots & 1 - \frac{a_{33}}{a_{33}+d_{33}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \\ -\frac{a_{n1}}{a_{nn}+d_{nn}} & & \cdots & & 1 - \frac{a_{nn}}{a_{nn}+d_{nn}} \end{pmatrix}. \quad (3.1)$$

Let R take the form

$$R = \begin{pmatrix} \frac{a_{11}}{a_{11}+d_{11}} & 0 & 0 & \dots & 0 \\ 0 & \frac{a_{22}}{a_{22}+d_{22}} & 0 & & \vdots \\ 0 & 0 & \frac{a_{33}}{a_{33}+d_{33}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \dots & \dots & & \frac{a_{nn}}{a_{nn}+d_{nn}} \end{pmatrix}. \tag{3.2}$$

Then, $B_{\ell_1} = R \cdot B + (I - R)$. Furthermore, R has the following property:

LEMMA 3.3. *If R takes the form (3.2) for a convergent Jacobi iteration process, and the matrix block decomposition fulfills $\theta > 1$ where $a_{ii} \geq \theta \sum_{j \in \Omega_0^{(i)}} |a_{ij}|$ for all $1 \leq i \leq n$, then for each element r_{ii} in R we have*

$$0 < r_{ii} \leq 1.$$

Proof. First, since we assume the Jacobi process to be convergent, $a_{ii} \neq 0\ \forall i$. Hence, $r_{ii} \neq 0$. Assuming $r_{ii} < 0$ we get from the definition of r_{ii} and $d_{ii} > 0$ by definition that $a_{ii} < 0$ and $d_{ii} > |a_{ii}|$, a contradiction to $\theta > 1$. Hence, $r_{ii} > 0$ and $d_{ii} < |a_{ii}|$ for $\theta > 1$. We then also have

$$|r_{ii}| = \left| \frac{a_{ii}}{a_{ii} + d_{ii}} \right| \leq \frac{|a_{ii}|}{|a_{ii} + d_{ii}|} \leq 1.$$

Furthermore we have $r_{ii} = 1 \Leftrightarrow d_{ii} = 0$. Hence, $0 < r_{ii} \leq 1$. □

Proof. To prove Theorem 3.2 we analyze the difference between $\bar{x} = 0$, the unique solution to (2.1) and the iterates x^t. For convenience, we assume for the right-hand side $c = 0$ in (2.1), and hence $d^\ell = 0$. Let $e^t = \bar{x} - x^t$ be the error in the t-th iteration. We consider the first $\bar{s}$ iterates in the process. Due to Theorem 2.2 there exists a positive v and $\alpha < 1$ such that $|B|\, v \leq \alpha v$. Since all components of v are positive, there exists a positive value M such that $|e^t| \leq Mv$ for $0 \leq t \leq \bar{s}$. We now consider any component i updated using any of these $\bar{s}$ vectors forming the first $\bar{s}$ iterates. Then, since we assumed $b = 0$ in (2.1) and hence $B_{\ell_1} x_i^t = B_{\ell_1} e_i^t$, the update satisfies

$$\begin{aligned} |e_i^{t+1}| &= \left| \sum_{j=1}^{n} b_{ij}^{\ell_1} e_j^{t-s(t,j)} \right| \\ &\leq \sum_{j=1}^{n} \left| r_{ii} b_{ij} e_j^{t-s(t,j)} \right| + \left| (1 - r_{ii}) e_i^{t-s(t,j)} \right| \\ &\leq |r_{ii}| \sum_{j=1}^{n} \left| b_{ij} e_j^{t-s(t,j)} \right| + |(1 - r_{ii})| \left| e_i^{t-s(t,j)} \right| \\ &\leq \alpha |r_{ii}| M v_i + |(1 - r_{ii})| M v_i \\ &\underbrace{=}_{0<r_{ii}\leq 1 \ \text{(Lemma 3.3)}} M v_i \left(\alpha r_{ii} + (1 - r_{ii}) \right) \end{aligned}$$

Let $\beta = 1 - r_{ii}(1 - \alpha)$. Then $0 \leq \beta < 1$ and

$$|e_i^{t+1}| \leq M v_i \left(\alpha r_{ii} + 1 - r_{ii} \right) = \beta M v_i.$$

Matrix name	$\#n$	$\#nnz$	cond(A)	cond($D^{-1}A$)	$\rho(M)$
CHEM97ZTZ	2,541	7,361	1.3e+03	7.2e+03	0.7889
FV1	9,604	85,264	9.3e+04	12.76	0.8541
FV3	9,801	87,025	3.6e+07	4.4e+03	0.9993
TREFETHEN_2000	2,000	41,906	5.1e+04	6.1579	0.8601

Table 4.1: Dimension and characteristics of the SPD test matrices and the corresponding iteration matrices.

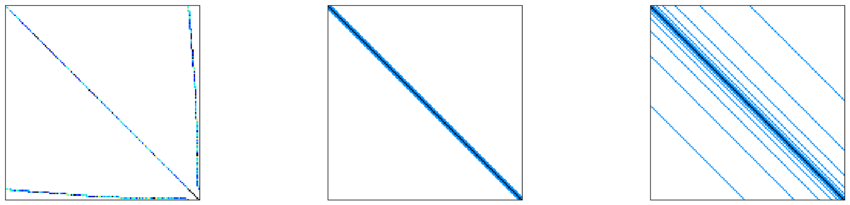

(a) CHEM97ZTZ (b) FV1, FV3 (c) TREFETHEN_2000

Fig. 4.1: Sparsity plots of test matrices.

If now t_1 is the first instance after $\bar{s}$ for which all components have been updated, then $|e^{t_1}| \leq \beta M v$. Moreover, $|e^t| \leq \beta M v$ for all $t \geq t_1$. Similarly, if t_2 is the next instance after t_1 for which all components have been updated a second time, then $|e_t| \leq \beta^2 M v$ for all $t \geq t_2$. This way we obtain that the error $|e^t| = |\bar{x} - x^t|$ converges to zero. Hence, the solution approximation is convergent to $\bar{x}$, the unique solution. $\square$

4. Experiment Setup.

4.1. Linear Equation Systems.

4.1. Linear Equation Systems. In our experiments, we search for the approximate solutions of linear systems of equations, where the respective matrices are taken from the University of Florida Matrix Collection (UFMC; see `http://www.cise.ufl.edu/research/sparse/matrices/`).

Due to the convergence properties of the iterative methods we analyze, the experiment matrices have to be chosen properly, fulfilling the sufficient convergence condition stated in section 2.1.

The matrix properties and sparsity plots can be found in Table 4.1 and Figure 4.1.

The first matrix, CHEM97ZTZ, comes from statistics [1]. Matrices FV1 and FV3 are finite element discretizations of the Laplace equation on a 2D mesh. Therefore they share a common sparsity structure, but differ in dimension and condition number due to the different finite element choice. The matrix TREFETHEN_2000 [29] is a 2000×2000 matrix where all entries are zero except for the ones at the positions (i, j) where $|i - j| = 2, 4, 8, 16 \ldots$. Furthermore, the main diagonal is filled with the primes $2, 3, 5, 7, 11 \ldots 17389$. Hence, this matrix has a lot off-diagonal entries, distributed over the diagonals that are by a power of 2 distant to the main diagonal.

[1]For more details see http://www.cise.ufl.edu/research/sparse/mat/Bates/README.txt

We furthermore take the number of right-hand sides to be one for all linear systems.

4.2. Hardware and Software Issues. The experiments were conducted on a heterogeneous GPU-accelerated multicore system located at the University of Tennessee, Knoxville. The system's CPU is one socket Intel Core Quad Q9300 @ 2.50GHz and the GPU is a Fermi C2050 (14 Multiprocessors x 32 CUDA cores @1.15GHz, 3 GB memory). The GPU is connected to the CPU host through a PCI-e$\times$16.

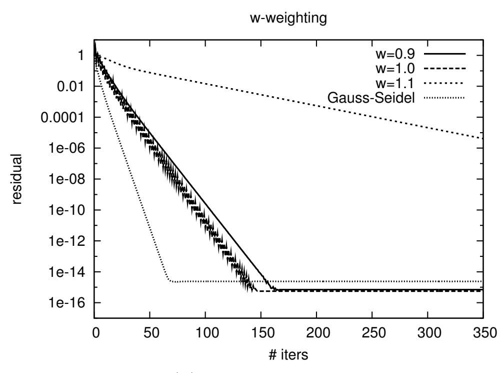
On the CPU, the synchronous Jacobi implementation runs on 4 cores. Intel compiler 11.1.069 [1] is used with optimization flag "-O3". The GPU implementation is based on CUDA [24], while the respective libraries used are from CUDA 4.0.17 [25]. The component updates were coded in CUDA, using thread blocks of size 512. The kernels are then launched through different streams. The thread block size, the number of streams, along with other parameters, were determined through empirically based tuning.

5. Numerical Experiments. In a first experiment, we analyze the influence of ω on the convergence rate with respect to global iteration numbers. Therefore we plot the relative residual depending on the iteration number. Note that all values are average due to the non-deterministic properties of block-asynchronous iteration. To have a reference, we additionally provide the convergence behavior of the sequential Gauss-Seidel algorithm.

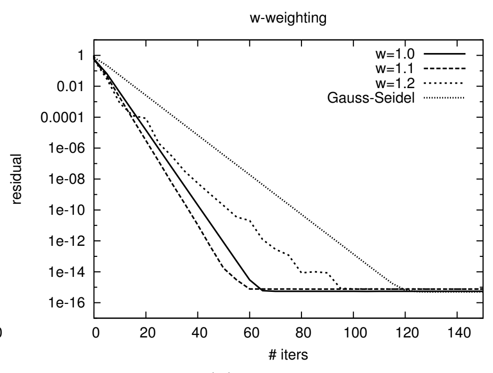
The results reveal, that the convergence rate of the block-asynchronous iteration is very dependent on the matrix characteristics. While for matrices with most relevant matrix entries gathered on or near the diagonal (small θ), the local iterations provide sufficient update improvement to overcompensate for the inferior convergence rate of the plain asynchronous iteration conducted globally. For these systems, e.g. FV1, and FV3 we achieve a higher convergence rate than the sequential Gauss-Seidel algorithm. Similar to the SOR algorithm, choosing $\omega > 1$ may improve the convergence further (Figure 5.1b and 5.1c). Considering the convergence rate of FV1 and FV3, we note that the condition number of the system has almost no influence.

For systems with considerable off-diagonal entries, the convergence of the block-asynchronous iteration decreases considerably (Figure 5.1a, 5.1d). The reason is that the off-diagonal entries are not taken into account for the local iterations. The purpose of the ℓ_1-weights introduced in section 3.2 is to account for these off-diagonal entries i.e. apply different weights in different rows. We now want to analyze whether applying them triggers convergence improvement. For these tests it is reasonable to choose a system with a high number of off-diagonal elements. Therefore, we will focus our analysis on the matrix TREFETHEN_2000 where, due to the design, the ratio between the diagonal entry and the offdiagonal parts differs considerably for the distinct rows. We now compare, using different block sizes, the convergence rate of the block-asynchronous iteration with the ℓ_1-weighted variant. Note at this point that using ℓ_1-weights triggers some overhead to the computation time due to the computation of the weights. This may be daunting for some systems where the convergence improvement is smaller than the overhead, but in most cases, the improved convergence rate directly correlates to a performance increase.

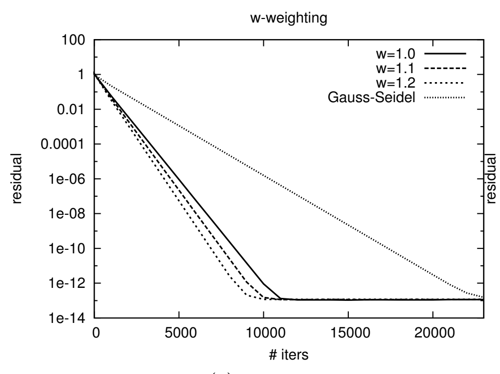
We can observe that, independent of the block size, using ℓ_1 weights improves the convergence rate. We also note that the influence of the block-size on the convergence rate for the unweighted algorithm is negligible. Using ℓ_1 weights is especially beneficial when targeting large block sizes, where the ℓ_1 weights for the distinct rows in one block



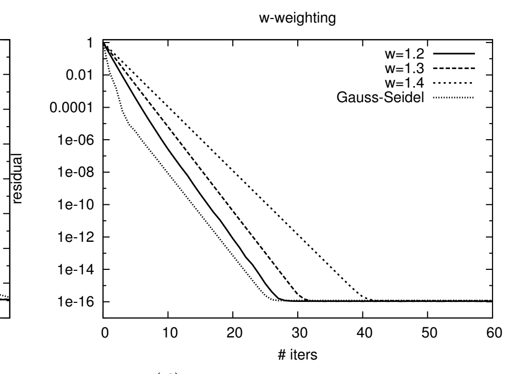
(a) Chem97ZtZ



(b) fv1



(c) fv3



(d) Trefethen_2000

Fig. 5.1: Convergence rate of ω-weighted block-asynchronous iteration.

differ considerably. For this case, the convergence of the block-asynchronous iteration is improved by a factor of almost 2.

While the convergence rate, with respect to iteration number, is interesting from the theoretical point of view, the more relevant factor is the performance. This depends not only on the convergence rate, but also on the efficiency of the respective algorithm on the available hardware resources. Whereas the Gauss-Seidel algorithm and the derived SOR algorithms require strict update order and therefore only allow sequential implementations, block-asynchronous iteration is very tolerant to update order and synchronization latencies, and therefore adequate for GPU implementations. In the next experiment, we analyze the time to solution for the the ω-weighted block-asynchronous iteration, and compare it with the SOR algorithm. Note at this point, that despite the similar notation, ω-weighting has, due to the algorithm design, a very different meaning in the SOR and the block-asynchronous iteration, respectively. For reasonable test cases, i.e. for the matrices with considerable off-diagonal entries (large θ), we provide the additional data for different ω-weights applied to the block-asynchronous algorithm enhanced by the ℓ_1-weighting technique.

The results show that for matrices where most entries are clustered on or near the main diagonal, the ω-weighted block asynchronous iteration outperforms the SOR method by more than an order of magnitude, see Figure 5.3b and 5.3c. Furthermore,

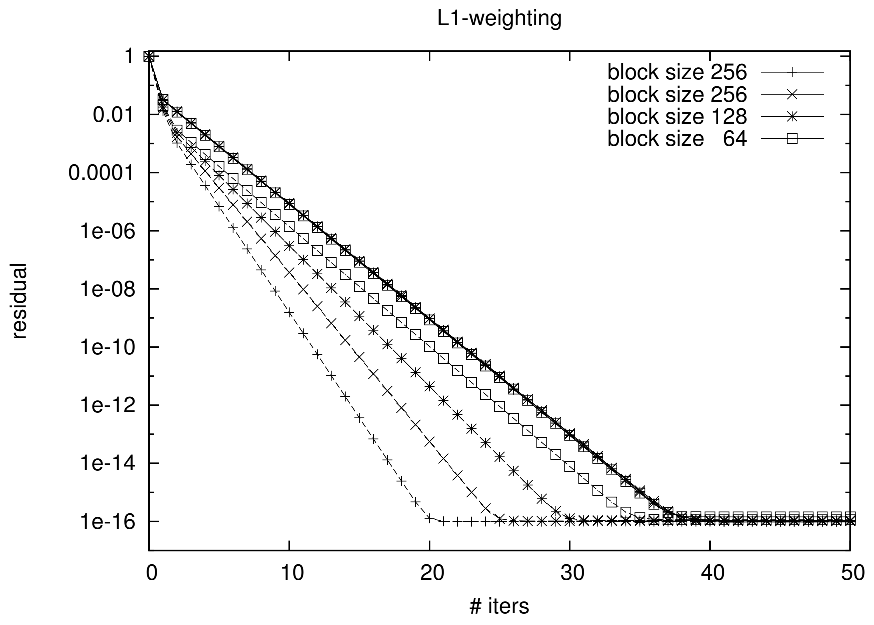


Fig. 5.2: Convergence improvement using ℓ_1 weights for different block sizes. Solid lines are block-asynchronous iteration, dashed lines are block-asynchronous iteration using ℓ_1 weights.

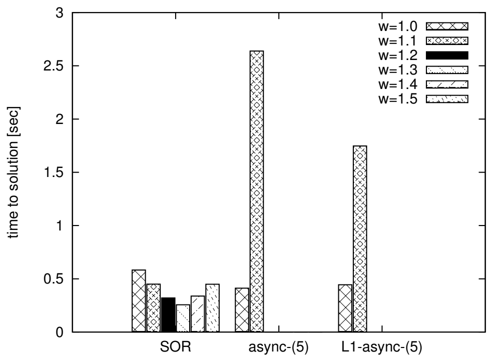
we observe, that ω-weights for the block-asynchronous algorithm have to be applied more carefully: already choosing $\omega \geq 1.4$ leads to divergence of all test cases. For matrices with considerable off-diagonal parts, using the block-asynchronous iteration may not pay off. Considering the runtime analysis for the matrix Chem97ZtZ (Figure 5.3a), we note that, although block-asynchronous iteration generates the solution faster than Gauss Seidel, the performance of the SOR algorithm cannot be achieved. Already, choosing $\omega = 1.1$ results in the loss of convergence. In fact, choosing $\omega \in [1, 1.1)$ may have positive effects, but due to the small problem size, the fast solution process and the overhead of the GPU kernel calls, no considerable performance gain can be expected.

For this case, the algorithm also does not benefit from enhancing it by ℓ_1-weights, since the overhead computing the ℓ_1 weights is large compared to the small off-diagonal part.

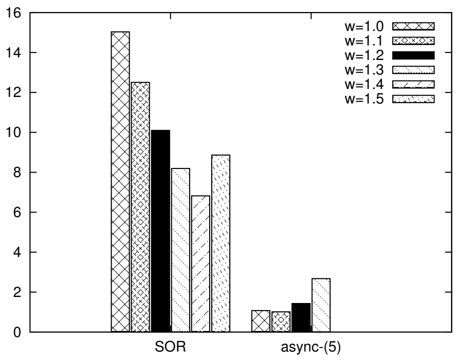
For the test matrix Trefethen_2000, the performance of SOR and block-asynchronous iteration is comparable for small ω. But enhancing the latter one with ℓ_1 weights triggers considerable performance increase. We then outperform the SOR algorithm by a factor of nearly 5. This reveals not only that using ℓ_1-weights pays off, but also the potential of applying a combination of both weighting techniques.

6. Conclusion.

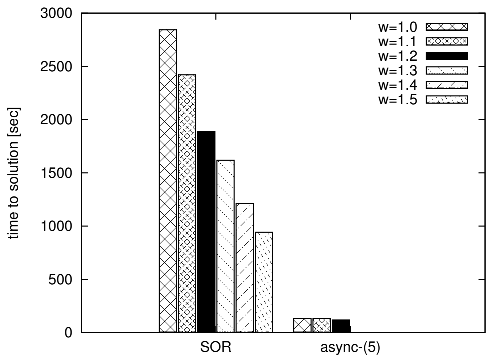
We introduced two weighting techniques for block-asynchronous iteration methods improving the performance and the convergence properties of the algorithm. Furthermore, we proved, that the sufficient convergence condition for the asynchronous iteration implies also the convergence of the weighted block-asynchronous iteration for adequate parameters. In numerical experiments with linear equation systems taken from the University of Florida Matrix collection, we were able to show



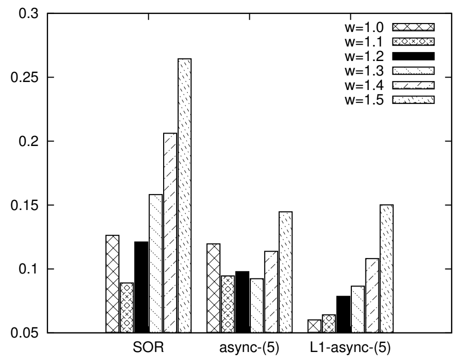
(a) Chem97ZtZ



(b) fv1



(c) fv3



(d) Trefethen_2000

Fig. 5.3: Time-to-solution comparison between SOR and weighted block-asynchronous iteration.

how the different techniques improve the convergence rate and the time-to-solution performance. The further research in this field will focus on how these improvements of weighted block-asynchronous methods impact multigrid methods. Especially for algebraic multigrid methods, where considerable off-diagonal entries are expected on the different grid levels, using weights for a block-asynchronous iteration smoother may be beneficial.

Acknowledgments. Implementation and computations were done at the Innovative Computing Laboratory at the University of Tennessee and the Engineering. The authors would like to thank the National Science Foundation, the Department of Energy, NVIDIA, and Microsoft Research for supporting this research effort.

REFERENCES

[1] Intel C++ Compiler Options. Intel Corporation. Document Number: 307776-002US.
[2] Hartwig Anzt, Stanimire Tomov, Jack Dongarra, and Vincent Heuveline. A block-asynchronous relaxation method for graphics processing units. Technical report, Innovative Computing Laboratory, University of Tennessee, UT-CS-11-687, 2011.
[3] Hartwig Anzt, Stanimire Tomov, Mark Gates, Jack Dongarra, and Vincent Heuveline. Block-

asynchronous Multigrid Smoothers for GPU-accelerated Systems. Technical report, Innovative Computing Laboratory, University of Tennessee, UT-CS-11-689, 2011.

[4] Ueresin Aydin and Michel Dubois. Generalized asynchronous iterations. pages 272–278, 1986.

[5] Ueresin Aydin and Michel Dubois. Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing*, 10(1):83–92, 1989.

[6] Roberto Bagnara. A unified proof for the convergence of jacobi and gauss-seidel methods. *SIAM Rev.*, 37:93–97, March 1995.

[7] Zhong-Zhi Bai, Violeta Migallón, José Penadés, and Daniel B. Szyld. Block and asynchronous two-stage methods for mildly nonlinear systems. *Numerische Mathematik*, 82:1–20, 1999.

[8] Allison H. Baker, Robert D. Falgout, Todd Gamblin, Tzanio V. Kolev, Schulz Martin, and Ulrike Meier Yang. Scaling algebraic multigrid solvers: On the road to exascale. *Proceedings of Competence in High Performance Computing CiHPC 2010*.

[9] Allison H. Baker, Robert D. Falgout, Tzanio V. Kolev, and Ulrike Meier Yang. Multigrid smoothers for ultra-parallel computing, 2011. LLNL-JRNL-435315.

[10] Gerard M. Baudet. Asynchronous iterative methods for multiprocessors. *J Assoc Comput Mach*, 25(2):226–244, 1978. cited By (since 1996) 86.

[11] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and distributed computation*. Prentice Hall, 1989.

[12] D. Chazan and W. Miranker. Chaotic Relaxation. *Linear Algebra and Its Applications*, 2(7):199–222, 1969.

[13] Michel Dubois and Faye A. Briggs. The run-time efficiency of parallel asynchronous algorithms. *IEEE Trans. Computers*, 40(11):1260–1266, 1991.

[14] Didier El Baz, Andreas Frommer, and Pierre Spiteri. Asynchronous iterations with flexible communication: contracting operators. *J. Comput. Appl. Math.*, 176(1):91–103, April 2005.

[15] M.N. El Tarazi. Some convergence results for asynchronous algorithms. *Numerische Mathematik*, 39(3):325–340, 1982.

[16] L. Elsner, I. Koltracht, and M. Neumann. On the convergence of asynchronous paracontractions with application to tomographic reconstruction from incomplete data. *Linear Algebra and its Applications*, 130(0):65 – 82, 1990.

[17] L. Elsner, I. Koltracht, and M. Neumann. Convergence of sequential and asynchronous nonlinear paracontractions. *Numerische Mathematik*, 62:305–319, 1992. 10.1007/BF01396232.

[18] Andreas Frommer. Generalized nonlinear diagonal dominance and applications to asynchronous iterative methods. *Journal of Computational and Applied Mathematics*, 38(1-3):105–124, 1991. cited By (since 1996) 8.

[19] Andreas Frommer and Hartmut Schwandt. Asynchronous parallel methods for enclosing solutions of nonlinear equations. *Journal of Computational and Applied Mathematics*, 60:47 – 62, 1995.

[20] Andreas Frommer and Daniel B. Szyld. Asynchronous two-stage iterative methods. *Numer. Math.*, 69(2):141–153, December 1994.

[21] Andreas Frommer and Daniel B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123:201–216, 2000.

[22] Boris Lubachevsky and Debasis Mitra. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *J. ACM*, 33:130–150, January 1986.

[23] Ulrike Meier Yang. On the use of relaxation parameters in hybrid smoothers. *Numerical Linear Algebra with Applications*, 11:155–172, 2011.

[24] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 2.3.1 edition, August 2009.

[25] NVIDIA Corporation. *CUDA TOOLKIT 4.0 READINESS FOR CUDA APPLICATIONS*, 4.0 edition, March 2011.

[26] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[27] John C. Strikwerda. A convergence theorem for chaotic asynchronous relaxation. *Linear Algebra and its Applications*, 253(1-3):15–24, March 1997.

[28] Daniel B. Szyld. Different models of parallel asynchronous iterations with overlapping blocks. *COMPUTATIONAL AND APPLIED MATHEMATICS*, 17:101–115, 1998.

[29] Nick Trefethen. Hundred-dollar, hundred-digit challenge problems. *SIAM News*, 35(1), January 2, 2002. Problem no. 7.

[30] R.S. Varga. *Matrix Iterative Analysis*. Springer Series in Computational Mathematics. Springer Verlag, 2010.