Noname manuscript No.
(will be inserted by the editor)

Performance Analysis of Asynchronous Parallel Jacobi

James Hook · Nicholas Dingle

the date of receipt and acceptance should be inserted later

Abstract The directed acyclic graph (DAG) associated with a parallel algorithm captures the partial order in which separate local computations are completed and how their outputs are subsequently used in further computations. Unlike in a synchronous parallel algorithm the DAG associated with an asynchronous parallel algorithm is not predetermined. Instead it is a product of the asynchronous timing dynamics of the machine and cannot be known in advance, as such it is best thought of as a pseudorandom variable.

In this paper we present a formalism for analyzing the performance of asynchronous parallel Jacobi's method in terms of its DAG. We use this approach to prove error bounds and bounds on the rate of convergence. The rate of convergence bounds are based on the statistical properties of the DAG and are valid for systems with a non-negative iteration matrix. We support our theoretical results with a suit of numerical examples, where we compare the performance of synchronous and asynchronous parallel Jacobi to certain statistical properties of the DAGs associated with the computations. We also present some examples of small matrices with elements of mixed sign, which demonstrate that determining whether a system will converge under asynchronous iteration in this more general setting is a far more difficult problem.

Keywords Asynchronous parallel Jacobi's method · chaotic iterations · parallel algorithm performance

School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK. (james.hook@manchester.ac.uk)

Numerical Algorithms Group Ltd, Oxford Street, Manchester, M1 5AN, UK. (nick.dingle@nag.co.uk)

1 Introduction

Jacobi's method for solving a linear system of equations iterates an affine map $x \mapsto Mx + c$. In a distributed memory parallel environment x, M and c can be broken up into N blocks of rows, so that processor i stores x_i, c_i and $M_{i:}$. Processor i is then responsible for updating its own block of rows and iterates the map by

$$x_i \mapsto M_{ii}x_i + c_i + \sum_{j \neq i} M_{ij}x_j,$$

where the x_j are the input from other processors. In order to update its state processor i must have appropriate input from its neighboring processors.

In the synchronous implementation, a processor cannot finish computing its update until it receives updates from its neighbors' previous computations:

```
for k=1,2,...
    x[i](k)=M[ii]*x[i](k-1)+c[i]
    wait to receive all x[j](k-1) values
    for all inputs j
        x[i](k)=x[i](k)+M[i][j]*x[j](k-1)
    end
    if converged stop
    broadcast new state x[i](k)
end
```

Delays in interprocessor communication give rise to delays and idle time in the whole computation. Introduced by Chazan and Miranker [6], chaotic or asynchronous iterations are parallel asynchronous processes in which idle time is removed by allowing individual processors to begin a new iteration as soon as their previous one is completed. If the processor requires input from a neighbor that it has not yet received, then it simply uses the most up-to-date value from that neighbor that it has available. This means that processors may use iterate components that are out-of-date and that some processors will be able to update many more times than others.

In asynchronous parallel Jacobi's method (APJ), processors successively update without waiting for communication from their neighbors:

```
for k=1,2,...
    x[i](k)=M[ii]*x[i](k-1)+c[i]
    for all inputs j
        x[i](k)=x[i](k)+M[i][j]*x[i,j]
    end
    if converged stop
    broadcast new state
end
```

Here `x[i,j]` represents processor i's most recently received update from j. The idea is that by minimizing idle time the update rate of the individual processors is increased and that this can improve the rate of convergence.

However the possible use of older input data can slow convergence and even lead to instability.

In [3] the authors implement synchronous and asynchronous parallel Jacobi using three different programming paradigms (MPI, SHMEM and OpenMP) and analyze the performance of these implementations using HECToR, the UK's national supercomputing service. They found that on large numbers (thousands) of cores asynchronous parallel Jacobi was approximately 10% faster than the synchronous method, and that the asynchronous method retained its performance in the presence of defects in the underlying hardware (a slow-running CPU) while the synchronous implementation did not.

In [7] the authors apply a variant of APJ where processors exchange updates every 5 iterations to the calculation of steady-state probabilities in large Markov chains. They also combine APJ with a matrix reordering scheme that minimizes the number of vector elements that need to be broadcast during each update. The combination of the two approaches is shown to yield better parallel speedups than an asynchronous Jacobi implementation that uses a naive matrix-partitioning approach running on the same hardware.

In [3, 5, 7] the authors study the performance of APJ in terms of time to converge. In this paper we will make a similar study except that we record a complete log of all of the iterations and communications that take place in the implementation. We record all data of the form "at time t processor i updates for the kth time using the mth value of processor j as an input".

The composite of all this data is a *directed acyclic graph* (DAG), whose vertices are the successively computed values associated with each processor and whose edges represent use in computation. The DAG contains an edge from processor j's mth update to processor i's kth update if processor i updates for the kth time using the mth value computed by processor j for its input. The DAG therefore gives us a complete account of the asynchronous dynamics of the algorithm's execution. We present upper and lower bounds on the rate of convergence of APJ that are based on statistical properties of the DAG, namely the update rate of the slowest processor and the shortest path growth rate. Note that although we think of the DAG as a pseudorandom variable and study some of its statistical properties, we do not actually model the DAG as a random variable or use any probability theory in our analysis. The exact structure of the DAG is determined by the timing of the different events taking place during the execution of the algorithm and can be very complicated. If we run the same experiment twice we can not expect to see exactly the same DAG. However, statistical measures like the ones we use in our bounds should be nearly identical between repeated runs of the same experiment. Developing probabilistic models of the parallel computation to predict these DAG statistics could be an interesting topic for future research.

The textbook of Bertsekas and Tsitsiklis [2] reviews many asynchronous methods for solving fixed point problems. For example computing the invariant distribution of a Markov chain. The authors prove a lower bound for the rate of convergence of this process, which is based on certain assumptions about the DAG, namely that the update rates are uniform and constant and that communicated data can only arrive up to a fixed number of iterations late. Their model labels individual communication channels as slow or fast. Data communicated along fast channels always arrives at most b iterations out-of-date and data communicated along slow channels at most B. Their lower bound on the rate of convergence can be thought of as a combination of two individual bounds, one for the slow component of the process and one for the fast component. The bounds on the slow/fast components are essentially the same as our shortest path performance Guarantee bound, except that rather than being based on the true shortest path growth rate, they are based on a bound on the shortest path growth rate, which is implied by the rule that says that slow/fast communicated data is never more than B/b iterations out-of-date respectively. A more recent review paper by Frommer and Szyld is available [9], the references of which contain many diverse applications of asynchronous linear solvers. Unlike the previous literature on asynchronous methods, we want to compare the rate of convergence of the asynchronous process to measurable statistics of the DAG, as opposed to parameters of some idealized model of the DAG, such as the maximum out-of-datedness parameter used in the bound in [2].

Asynchronous linear system solvers are not limited to Jacobi's method. Avron, Druinsky and Gupta [1] propose a shared-memory asynchronous method for general symmetric positive definite matrices based on the randomized variant of the Gauss-Seidel iteration. In fact any variant of the Gauss-Seidel iteration will admit the formulation we develop in Section 2. The classical Gauss-Seidel iteration or over-relaxation methods give rise to an iteration with a predetermined DAG, whilst randomized Gauss-Seidel iteration gives rise to a random DAG whose randomness derives from a pseudorandom number generator rather than the chaotic dynamics of the execution itself (as is the case in APJ).

Some more general asynchronous parallel iterative algorithms include the following. Niu et al [15] present `Hogwild!`, which is an asynchronous stochastic gradient descent algorithm. Elsener, Koltracht and Neumann present a shared memory asynchronous implementation of Kaczmarz method for solving linear equations [8], which more recently has been restudied in the distributed memory setting by Liu, Wright and Sridhar [13]. Lu and Tang [14] propose an algorithm for solving systems with symmetric positive definite matrices over a network of asynchronous agents, whose interactions are controlled by some (random) external network dynamics. For example the agents could be thought of as devices in a wireless network which are physically moving around, so that their network of connectivity is constantly changing. Several of these papers also provide performance bounds which are based on idealized assumptions (such as maximum out-of-datedness) about the DAG. It may be possible to adapt the results we present in Section 3 to these different algorithms, to provide performance bounds based on statistical properties of the actual DAG instead.

This paper is organized as follows. In Section 2 we review APJ, introduce all of our important definitions and prove error bounds for the asynchronous iteration. In Section 3 we prove a sandwich of bounds on the rate of convergence of APJ for systems with non-negative iteration matrix. In section 4 we make some remarks about the general case (matrices with elements of mixed sign) and give some simple examples exhibiting interesting behaviors. In Section 5 we present a suite of numerical experiments in which we run APJ on a multicore machine and make a precise log of the DAG during the algorithm's execution.

2 Asynchronous Parallel Jacobi

For $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$ Jacobi's method attempts to solve the linear system $Ax = b$ by computing the sequence of vectors $x(k)$ $k = 0, 1, \dots$ by

$$x(k) = Mx(k-1) + c, \quad M = -D^{-1}(L+U), \quad c = D^{-1}b, \tag{1}$$

where D, L and U are the diagonal and strictly lower and upper triangular parts of A respectively. We call $x(k)$ the kth *update* and we call M the *iteration matrix.* In the synchronous version of Jacobi's method the sequence of updates converge to the solution of $Ax = b$ as $n \to \infty$ if $\rho(M) < 1$, where $\rho(M)$ is the *spectral radius* of M. The *rate of convergence* of synchronous Jacobi's method is equal to

$$\lambda = -\alpha \log\big(\rho(M)\big),$$

where α is the *update rate*, that is the average number of updates computed per second. Thus provided $\rho(M) < 1$ and $\alpha > 0$, the synchronous scheme will converge exponentially quickly as $t \to \infty$.

In a parallel synchronous implementation of Jacobi's method the vectors x and c are partitioned into N blocks of coordinates and the matrix M is partitioned into $N \times N$ block submatrices. We denote these blocks using subscripts so that x_i is the ith block of x coordinates and M_{ij} is the (i, j)th block submatrix of M. Processor i is then responsible for the x_i and iteratively computes the sequence of vectors $x_i(1), x_i(2), \dots$ using

$$x_i(k) = \sum_{j=1}^{N} M_{ij} x_j(k-1) + c_i. \tag{2}$$

We call $x_i(k)$ the kth *update* of processor i.

Jacobi's method is also amenable to an asynchronous parallel implementation in which newly-computed vector updates are exchanged when they become available rather than by all processors at the end of each iteration. In APJ processor i iteratively computes the sequence of vectors $x_i(1), x_i(2), \dots$ using

$$x_i(k) = \sum_{j=1}^{N} M_{ij} x_j\big(\Psi(i, j, k)\big) + c_i, \tag{3}$$

where $\Psi(i, j, k) = m$ if processor i uses processor j's mth update in the computation of its kth update. We call $\Psi : \{1, \dots, N\}^2 \times \mathbb{N} \mapsto \mathbb{N}$ the *input index.* In an asynchronous algorithm the input index cannot be known a priori so is best thought of as a pseudorandom variable, which is determined by the exact timings of the different computations and communications taking place in the machine during the execution of the algorithm

The *directed acyclic graph* (DAG) associated with Ψ is the directed graph with vertices given by the processor updates $\{x_i(k) : i = 1, \dots, N : k = 1, 2, \dots\}$ and with a directed edge from $x_j(m)$ to $x_i(k)$, whenever $\Psi(i, j, k) = m$. This rather general formulation can be used to describe the case of synchronous parallel Jacobi's method by setting $\Psi(i, j, k) = k - 1$ for all i, j, k. This corresponds to a highly regular DAG in which each processor update receives an input from every processor update of the previous iteration.

Iteration matrices of interest will often be sparse and have a sparse block structure. If a block submatrix is equal to zero then we can ignore it in our formulation of the update equations (3) as well as in the recording of Ψ and the construction of the DAG. So that if $M_{ij} = 0$ then there will be no direct communication from processor j to processor i and we therefore do not include any edges from processor j updates to processor i updates in the DAG.

Let $\nu_i(t)$ record the total number of updates performed by processor i by time t. The state of the system at time t, denoted $x[t]$, is given by the direct product of each of the processor's most recently computed values

$$x[t] = \Big(x_1\big(\nu_i(t)\big), \dots, x_N\big(\nu_N(t)\big)\Big).$$

2.1 Decoupling error and solution

For synchronous Jacobi's method it is easy to see that the solution can be decoupled from the error in the update vectors and that the error then evolves according to a linear (rather than affine) update rule. Let $x(*)$ be the unique solution to $Ax = b$ and set $x(k) = x(*) + y(k)$ then

$$\begin{aligned} x(*) + y(k) &= M\big(x(*) + y(k-1)\big) + c \\ &= Mx(*) + c + My(k-1) \\ &= x(*) + My(k-1), \end{aligned}$$

so that the *error vectors* $y(0), y(1), \dots$ evolve according to

$$y(k) = My(k-1). \tag{4}$$

If the iteration matrix M has spectral radius less than one, then the error terms will converge exponentially to zero as $k \to \infty$ and the update vectors will converge to the solution. Otherwise, if the iteration matrix has spectral radius greater than one then generically[1] the errors will grow exponentially and the update vectors will diverge.

The solution and the error can also be decoupled in asynchronous Jacobi's method. As before we set $x_i(k) = x(*)_i + y_i(k)$. The update rule for processor i becomes

$$\begin{aligned} x_i(*) + y_i(k) &= \sum_{j=1}^{N} M_{ij}\Big(x_j(*) + y_j\big(\Psi(i,j,k)\big)\Big) + c_i \\ &= x_i(*) + \sum_{j=1}^{N} M_{ij} y_j\big(\Psi(i,j,k)\big), \end{aligned}$$

so that the error vectors $y_i(0), y_i(1), \ldots$ evolve according to the linear update rule

$$y_i(k) = \sum_{j=1}^{N} M_{ij} y_j\big(\Psi(i,j,k)\big). \tag{5}$$

If the error vectors converge to zero as $k \to \infty$ then the update vectors will converge to the solution. Otherwise, if the errors diverge then so too will the update vectors.

We define the *error operator*

$$\Omega(\Psi, M, t) : x[0] - x(*) \mapsto x[t] - x(*), \tag{6}$$

which maps the initial error $x(0) - x(*) = y(0)$ to the error at time t, which is given by

$$x[t] - x(*) = y[t] = \Big(y_1\big(\nu_1(t)\big), \ldots, y_N\big(\nu_N(t)\big)\Big).$$

To compute $y[t]$ we must compute all of the errors $y_i(1), \ldots, y_i\big(\nu_i(t)\big)$ for each $i = 1, \ldots, N$, from the initial errors $y_i(0)$, $i = 1, \ldots, N$, which can be done through repeated application of (5).

2.2 Paths through the DAG

Let $P(i,j,k)$ be the set of all paths through the DAG from $x_j(0)$ to $x_i(k)$, i.e. from processor j's initial condition to processor i's kth update. We represent paths as sequences of vertices

$$\sigma = \big(x_{i(0)}(k_0), x_{i(1)}(k_1), \ldots, x_{i(\ell)}(k_\ell)\big),$$

and we have $\sigma \in P(i,j,k)$ if and only if $i(0) = j$, $k_0 = 0$, $i(\ell) = i$, $k_\ell = k$ and $\Psi\big(i(m), i(m-1), k_m\big) = k_{m-1}$ for $m = 1, \ldots, \ell$. The first four conditions ensure

[1] If the initial error term $y(0)$ lies in an invariant subspace S, such that M restricted to S has spectral radius less than one then the error vector should converge to zero, although this behavior may be extremely sensitive to numerical error.

that σ starts at $x_j(0)$ and ends at $x_i(k)$, the fifth condition ensures that each transition in σ corresponds to an edge in the DAG. For a path $\sigma \in P(i,j,k)$ we define the *length* to be the number of transitions $L(\sigma) = \ell$ and the *weight* to be the product of the sequence of block submatrices corresponding to those transitions

$$W(\sigma) = M_{i(\ell),i(\ell-1)} \times \cdots \times M_{i(2),i(1)} \times M_{i(1),i(0)}.$$

Theorem 1 *The (i,j)th block submatrix of the error operator is the sum of the weights of all of the paths through the DAG from processor j's initial condition to the most recent update of processor i*

$$\Omega(\Psi, M, t)_{ij} = \sum_{\sigma \in P\big(i,j,\nu_i(t)\big)} W(\sigma).$$

Proof First we prove by induction that

$$y_i(k) = \sum_{j=1}^{N} \sum_{\sigma \in P(i,j,k)} W(\sigma) y_j(0), \tag{7}$$

for all $i = 1,\ldots,N$, $k = 1,2,\ldots$ Assume that the condition is satisfied for every update that occurs before some time t. Now consider the next update to occur, $x_i(k)$ say. The error at the update is given by

$$y_i(k) = \sum_{r=1}^{N} M_{ir} y_r\big(\Psi(i,r,k)\big), \tag{8}$$

where each of the $y_r\big(\Psi(i,r,k)\big)$, $r = 1,\ldots,N$ satisfy (7) by the induction hypothesis.

Let $\sigma \in P\big(r,j,\Psi(i,r,k)\big)$ be a path through the DAG from $x_j(0)$ to $x_r\big(\Psi(i,r,k)\big)$ then σ can be extended to create a path from $x_j(0)$ to $x_i(k)$ with

$$\sigma' = \big(\sigma, x_i(k)\big) \in P(i,j,k), \quad W(\sigma') = M_{i,r} W(\sigma).$$

Clearly every path through the DAG from an initial condition to $x_i(k)$ arises in this way. Substituting (7) into (8), then identifying terms of the form $M_{i,r}W(\sigma)$ as being the weights of extended paths $\sigma' \in P(i,j,k)$ we obtain

$$\begin{aligned} y_i(k) &= \sum_{r=1}^{N} M_{i,r} \sum_{j=1}^{N} \sum_{\sigma \in P\big(r,j,\Psi(i,r,k)\big)} W(\sigma) y_j(0) \\ &= \sum_{j=1}^{N} \sum_{r=1}^{N} \sum_{\sigma \in P\big(r,j,\Psi(i,r,k)\big)} M_{i,r} W(\sigma) y_j(0) \\ &= \sum_{j=1}^{N} \sum_{\sigma' \in P(i,j,k)} W(\sigma') y_j(0). \end{aligned}$$

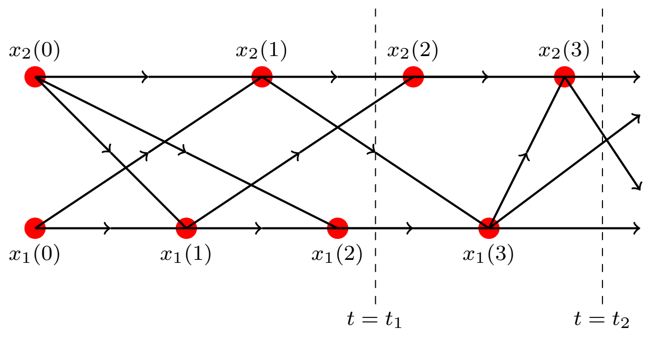


Fig. 1: Example 1. Sample of DAG.

Finally note that

$$\Omega(\Psi, M, t)_{ij} = \frac{dy_i\big(\nu_i(t)\big)}{dy_j(0)} = \sum_{\sigma \in P\left(i,j,\nu_i(t)\right)} W(\sigma). \ \square$$

Example 1 Suppose that we are iterating the map $x \mapsto Mx + c$ on a two processor machine. A sample of a possible input index and corresponding DAG are displayed in Table 1 and Figure 1.

Table 1: Example 1. Sample of input index Ψ

k	$\Psi(1,1,:)$	$\Psi(1,2,:)$	$\Psi(2,1,:)$	$\Psi(2,2,:)$
1	0	0	0	0
2	1	0	1	1
3	2	1	3	2

If the initial state in processor 1 is $x_1(0)$ and the initial state in processor 2 is $x_2(0)$ then processor 1's first update is given by

$$x_1(1) = M_{1,1}x_1(0) + M_{1,2}x_2(0) + c_1,$$

processor 2's first update is given by

$$x_2(1) = M_{2,1}x_1(0) + M_{2,2}x_2(0) + c_2,$$

processor 1's second update by

$$\begin{aligned} x_1(2) &= M_{1,1}x_1(1) + M_{1,2}x_2(0) + c_1 \\ &= M_{1,1}(M_{1,1}x_1(0) + M_{1,2}x_2(0) + c_1) + M_{1,2}x_2(0) + c_1, \end{aligned}$$

and so on.

The error operator can be computed either by repeated application of (5) or by summing over all of the paths though the DAG according to Theorem 1. For example

$$\Omega(\Psi, M, t_1) = \begin{bmatrix} M_{1,1}M_{1,1} & M_{1,2} + M_{1,1}M_{1,2} \\ M_{2,1} & M_{2,2} \end{bmatrix},$$

and

$$\Omega(\Psi, M, t_2) = \begin{bmatrix} M_{1,1}M_{1,1}M_{1,1} + M_{1,2}M_{2,1} & M_{1,1}M_{1,2} + M_{1,1}M_{1,1}M_{1,2} \\ & +M_{1,2}M_{2,2} \\ & \\ M_{2,2}M_{2,2}M_{2,1} + M_{2,2}M_{2,1}M_{1,1} & M_{2,2}M_{2,2}M_{2,2} + M_{2,1}M_{1,2}M_{2,2} \\ +M_{2,1}M_{1,1}M_{1,1}M_{1,1} + M_{2,1}M_{1,2}M_{2,1} & +M_{2,1}M_{1,1}M_{1,2} + M_{2,2}M_{2,1}M_{1,2} \\ & +M_{2,1}M_{1,1}M_{1,1}M_{1,2} \end{bmatrix}.$$

We verify Theorem 1 for the $(1,2)$ block entry of the error operator at time t_2

$$\begin{aligned} \Omega(\Psi, M, t_2)_{1,2} &= \sum_{\sigma \in P(1,2,3)} W(\sigma) \\ &= W\big(x_2(0), x_1(2), x_1(3)\big) + W\big(x_2(0), x_1(1), x_1(2), x_1(3)\big) \\ &\quad + W\big(x_2(0), x_2(1), x_1(3)\big) \\ &= M_{1,1}M_{1,2} + M_{1,1}M_{1,1}M_{1,2} + M_{1,2}M_{2,2}. \end{aligned}$$

2.3 Stopping conditions

The standard stopping condition for parallel synchronous Jacobi's method is

$$\|D_{ii}r_i(k)\|_\infty \leq \epsilon\|b\|, \quad \text{for } i = 1, \dots, N, \tag{9}$$

where $r_i(k) = x_i(k) - x_i(k+1)$ and where $\epsilon > 0$ is the required precision. Suppose that condition (9) is satisfied, then we have

$$x(k) = Mx(k) + c + r(k) \Leftrightarrow Ax(k) = b + Dr(k),$$

so that

$$\frac{\|x(*) - x(k)\|}{\|x(*)\|} \leq \frac{\kappa(A)\|Dr(k)\|}{\|b\|} \leq \epsilon\kappa(A)\sqrt{n}, \tag{10}$$

where $\kappa(A)$ is the 2-norm condition number of A, which is equal to the ratio of A's largest and smallest singular values. This bound reflects the relationship between the residual and forwards error which is standard for all numerical linear systems solvers.

In our implementation of asynchronous Jacobi's method each processor checks its local stopping condition (9) as before. When a processor reaches its stopping condition it broadcasts this fact up a communication tree to the root processor which is responsible for checking all of the other processors' stopping conditions. When all of the processors have met their stopping conditions the root processor stops iterating itself and broadcasts an instruction back down the tree telling the other processors to stop. As a result processors will typically continue iterating for a small number of steps after they have reached their local stopping condition. In the error analysis below we will make the following assumptions, suppose that the asynchronous iteration stops at time t, then

1. At time t processor i's most recent $\eta_i > 0$ iterations
$$x_i\big(\nu_i(t) - \eta_i + 1\big), \dots, x_i\big(\nu_i(t)\big),$$
all satisfy the local stopping condition, for $i = 1, \dots, N$. Recall that the local iteration number of processor i at time t is denoted $\nu_i(t)$.
2. At time t any input from processor j to processor i's most recent update comes from an iteration that satisfied its local stoping condition
$$\Psi\big(i, j, \nu_i(t)\big) \geq \nu_j(t) - \eta_j + 1, \quad i, j = 1, \dots, N.$$

It does not follow that solutions obtained using our stopping protocol will necessarily satisfy the above assumptions. This is because it is possible to construct iteration matrices, such that under synchronous iteration, each processor periodically satisfies its local stopping condition long before global convergence is achieved. While it is possible that the randomness introduced by asynchronicity can make this behavior less likely to manifest itself during asynchronous Jacobi's method, it must still be properly guarded against for a stopping condition with a guaranteed error bound. Presently no reliable and truly asynchronous method for stopping has been developed and this is an important topic for future research. In practice when using asynchronous Jacobi's method, we would recommend using our simple asynchronous stopping protocol followed by one iteration of synchronous Jacobi, in order to get a guaranteed check for convergence. If the synchronous step does not report convergence, then the asynchronous iteration should be restated, or the remainder of the solve could be run synchronously. In the numerical experiments that we present in Section 5, we only use our simple asynchronous stopping protocol, in spite of its potential unreliability. After each computation is finished we are able to interrogate the data collected during the asynchronous solves and can confirm that the solutions which we obtain always satisfy the above assumptions.

Suppose that the asynchronous iteration is stopped at time t. Then the final iterates are given by

$$\begin{aligned} x_i\big(\nu_i(t)\big) &= \sum_{j=1}^{N} M_{ij} x_j\Big(\Psi\big(i, j, \nu_i(t)\big)\Big) + c_i \\ &= \sum_{j=1}^{N} M_{ij}\Big(x_j\big(\nu_j(t)\big) + f_{ij}\Big) + c_i, \quad \text{for } i = 1, \dots, N, \end{aligned} \tag{11}$$

where $f_{ij} = x_j\Big(\Psi\big(i, j, \nu_i(t)\big)\Big) - x_j\big(\nu_j(t)\big)$ is the difference between processor j's local state at time t and the value that processor i used for processor j's state in its most recent update. Expanding (11) and using $M_{ij} = -D_{ii}^{-1}A_{ij} + \delta_{ij}I$ gives

$$x_i\big(\nu_i(t)\big) = -\sum_{j=1}^{N} D_{ii}^{-1}A_{ij}x_j\big(\nu_j(t)\big)+x_i\big(\nu_i(t)\big)+\sum_{j=1}^{N} M_{ij}f_{ij}+c_i, \quad \text{for } i = 1,\dots,N.$$

Rearranging using $\sum_{j=1}^{N} A_{ij}x_j\big(\nu_j(t)\big) = \big(Ax[t]\big)_i$ and $D_{ii}c_i = b_i$ gives

$$(Ax[t])_i = b_i + \sum_{j=1}^{N} D_{ii}M_{ij}f_{ij}, \quad \text{for } i = 1,\dots,N.$$

Therefore the relative forwards error satisfies

$$\frac{\|x[t] - x(*)\|}{\|x(*)\|} \le \frac{\kappa(A)\sqrt{\sum_{i=1}^{N} \| \sum_{j=1}^{N} D_{ii}M_{ij}f_{ij}\|^2}}{\|b\|}, \tag{12}$$

where

$$\begin{aligned}\sqrt{\sum_{i=1}^{N} \| \sum_{j=1}^{N} D_{ii}M_{ij}f_{ij}\|^2} &= \sqrt{\sum_{i=1}^{N} \| \sum_{j=1}^{N} D_{ii}M_{ij}D_{jj}^{-1}D_{jj}f_{ij}\|^2} \\ &\le \sqrt{\sum_{i=1}^{N} \| \sum_{j=1}^{N} |D_{ii}M_{ij}D_{jj}^{-1}||D_{jj}f_{ij}|\|^2}. \end{aligned} \tag{13}$$

The difference between processor j's local state at time t and the value that processor i used for processor j's state in its most recent update satisfies

$$\begin{aligned} f_{ij} &= x_j\Big(\Psi\big(i,j,\nu_i(t)\big)\Big) - x_j\big(\nu_j(t)\big) \\ &= \sum_{k=\Psi\big(i,j,\nu_i(t)\big)}^{\nu_j(t)-1} x_j(k) - x_j(k+1) = \sum_{k=\Psi\big(i,j,\nu_i(t)\big)}^{\nu_j(t)-1} r_j(k), \end{aligned}$$

and from assumptions 1 and 2, we have $\|D_{jj}r_j\|_\infty \le \epsilon\|b\|$ for $\Psi\big(i,j,\nu_i(t)\big) \le k \le \nu_j(t)$, so that

$$\|D_{jj}f_{ij}\|_\infty \le \epsilon\|b\|k_{ij}, \tag{14}$$

where $k_{ij} = \nu_j(t)-\Psi\big(i,j,\nu_i(t)\big)$ is the difference between processor j's iteration number and the iteration number of the value that processor i is using for processor j's state. Therefore

$$|D_{jj}f_{ij}| \le \underline{1}_{n/N}\epsilon\|b\|k_{\max}, \quad \text{for } j = 1,\dots,N,$$

where $\underline{1}_{n/N}$ is an n/N-vector of ones and $k_{\max} = \max_{ij} k_{ij}$. Substituting into (13) gives

$$\sqrt{\sum_{i=1}^{N} \| \sum_{j=1}^{N} D_{ii} M_{ij} f_{ij} \|^2} \leq k_{\max} \epsilon \|b\| \sqrt{\sum_{i=1}^{N} \| \sum_{j=1}^{N} |D_{ii} M_{ij} D_{jj}^{-1}| \underline{1}_{n/N} \|^2}$$
$$\leq \epsilon \|b\| k_{\max} \left\| |AD^{-1}| \underline{1}_n \right\| = k_{\max} \epsilon \|b\| \left\| |AD^{-1}| \right\|_{\infty,2},$$

which results in the bound

$$\frac{\|x[t] - x(*)\|}{\|x(*)\|} \leq \epsilon \kappa(A) k_{\max} \left\| |AD^{-1}| \right\|_{\infty,2}, \tag{15}$$

where $\left\| |AD^{-1}| \right\|_{\infty,2}$ is the $(\infty, 2)$ norm of the absolute value of AD^{-1}. Note that if A is columnwise diagonal dominant then we have $\left\| |AD^{-1}| \right\|_{\infty,2} \leq 2\sqrt{n}$.

In our experiments we find that $k_{\max}$ is never greater than 5, so that our bound guarantees that the error in the asynchronous solution will be the same order of magnitude as the error in the synchronous solution. In our experiments on non-negative matrices we find that the synchronous and asynchronous iterations return approximate solutions with near identical residuals and errors. In fact the addition iterations performed between the processors first meeting their stopping conditions and then receiving their stopping instruction, result in the asynchronous iteration returning a solution with slightly smaller error than the synchronous iteration with the same precision ϵ.

3 Theory for non-negative matrices

If the iteration matrix M is non-negative, i.e. $m_{ij} \in \mathbb{R}_+$ for all $i, j = 1, \ldots, n$, then we are able to prove exponential convergence (under very mild assumptions about the DAG) and provide lower and upper bounds for the rate of convergence. The necessary and sufficient condition on the problem matrix A for the iteration matrix M to be non-negative is that every off diagonal non-zero entry a_{ij} must have the opposite sign to the diagonal entry in its row a_{ii}. This condition is satisfied by any graph laplacian matrix and therefore by the matrix discretization of any PDE system whose dynamics are dominated by a diffusion term.

The following results will be useful in our analysis.

Lemma 1 (Properties of operator norm) *Let*

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2},$$

denote the l_2 operator norm.

1. *If A is a non-negative matrix and B is a submatrix of A then $\|A\| \geq \|B\|$.*
2. *If A and B are non-negative $d \times d$ matrices then $\|A+B\| \geq \max\{\|A\|, \|B\|\}$.*

3. $\|AB\| \leq \|A\|\|B\|$.
4. $\|A\| \geq \rho(A)$.
5.

$$\lim_{k\to\infty} \frac{\log \|A^k\|}{k} = \log\big(\rho(A)\big),$$

6. If $\rho(A) < 1$ *then there exists* $C \in \mathbb{R}$ *such that* $1 \leq \max_{k=0}^{\infty} \|A^k\| \leq C$.
7. If $\rho(A) > 1$ *then*

$$\lim_{k\to\infty} \frac{\max_{m=1}^{k} \log \|A^m\|}{k} = \log\big(\rho(A)\big).$$

Proof Items 1, 2, 3 and 4 are trivial, item 5 follows by considering the Jordan form of A [12].

• Item 6. First note that $\|A^0\| = \|I\| = 1$ so that $1 \leq \max_{k=0}^{\infty} \|A^k\|$. It follows from item 5 that $\lim_{k\to\infty} \|A^k\| = 0$, so there exists $k_1 \in \mathbb{N}$, such that $\|A^k\| < 1$ for all $k \geq k_1$. Therefore $\max_{k=0}^{\infty} \|A^k\| = \max_{k=0}^{k_1} \|A^k\|$. Finally using item 3 we have

$$\max_{k=0}^{\infty} \|A^k\| \leq \max_{k=0}^{k_1} \|A\|^k = \max\{1, \|A\|^{k_1}\} = C.$$

• Item 7. From item 5, there exists k_ϵ such that

$$\frac{\log \|A^k\|}{k} \leq \rho(A) + \epsilon,$$

for all $k \geq k_\epsilon$. Therefore

$$\frac{\max_{m=1}^{k} \log \|A^m\|}{k} \leq \max\Big\{\frac{\max_{m=1}^{k_\epsilon} \log \|A^m\|}{k}, \rho(A) + \epsilon\Big\}.$$

Since $\rho(A) > 1$, item 4 implies $\|A\| > 1$. From item 3 we have

$$\frac{\max_{m=1}^{k_\epsilon} \log \|A^m\|}{k} \leq \frac{\max_{m=1}^{k_\epsilon} \log \|A\|^m}{k} = \frac{k_\epsilon}{k} \log \|A\|.$$

Finally for all $k \geq k_\epsilon/\epsilon$ we have

$$\frac{\max_{m=1}^{k} \log \|A^m\|}{k} \leq \max\big\{\epsilon \log \|A\|, \rho(A) + \epsilon\big\},$$

taking the limit $\epsilon \to 0$ we obtain

$$\lim_{k\to\infty} \frac{\max_{m=1}^{k} \log \|A^m\|}{k} = \rho(A). \ \square$$

Theorem 2 (Weakest link performance barrier) *Suppose that the problem* $Ax = b$ *has non-negative Jacobi iteration matrix* M, *which we iterate asynchronously with input index* Ψ. *Let* M_{ii} *be the block submatrix iterated by processor* i *and let* $\nu_i(t)$ *be the number of times that processor* i *has updated by time* t. *We have the bound*

$$\|\Omega(\Psi, M, t)\| \geq \|M_{ii}^{\nu_i(t)}\|, \tag{16}$$

so that for generic initial condition $x(0)$

$$\limsup_{t\to\infty} \frac{\log \|x[t] - x(*)\|}{t} \geq \max_i a_i^* \log\big(\rho(M_{ii})\big),$$

where

$$a_i^* = \lim_{t\to\infty} \frac{\nu_i(t)}{t},$$

is processor i's update rate.

Proof From Theorem 1 we have

$$\Omega(\Psi, M, t)_{ii} = \sum_{\sigma \in P\big(i,i,\nu_i(t)\big)} W(\sigma).$$

Recall that $P\big(ii, \nu_i(t)\big)$ is the set of all paths through the DAG from processor i's initial condition to its most recent update. One such path is given by

$$\sigma = \Big(x_i(0), x_i(1), \ldots, x_i\big(\nu_i(t)\big)\Big),$$

which has weight $W(\sigma) = M_{ii}^{\nu_i(t)}$. Therefore since all of the path weights are non-negative matrices, by Lemma 1 items 1 and 2, we have

$$\|\Omega(\Psi, M, t)\| \geq \|\Omega(\Psi, M, t)_{ii}\| \geq \|M_{ii}^{\nu_i(t)}\|.$$

For the second result we need to take the SVD of the error operator

$$USV^\top = \Omega(\Psi, M, t),$$

where U, S, V are parametrized by Ψ, M, t and the diagonal entries of S are ordered in decreasing modulus. Now

$$\begin{aligned} x[t] - x(*) &= \Omega(\Psi, M, t)\big(x[0] - x(*)\big) \\ &= U_1 S(1,1)\langle x[0] - x(*), V_1\rangle + \text{terms perpendicular to } U_1, \end{aligned}$$

and using the fact that $S(1,1) = \|\Omega(\Psi, M, t)\| \geq \|M_{ii}^{\nu_i(t)}\|$, we have

$$\|x[t] - x(*)\| \geq \|U_1 S(1,1)\langle x(0) - x(*), V_1\rangle\| \geq \|M_{ii}^{\nu_i(t)}\| \; |\langle x(0) - x(*), V_1\rangle|.$$

Since the SVD of the error operator is a function of t we need to be a bit careful here. To make the time-dependence explicit we now include the parameterization in the notation $V_1(t)$. Consider the set

$$K = \{x \in \mathbb{C}^n : \lim_{t\to\infty} \langle x, V_1(t)\rangle = 0\},$$

clearly this set forms a subspace of $\mathbb{C}^n$. Suppose that $K = \mathbb{C}^n$, then $\lim_{t\to\infty} V_1(t) = 0$, which is a contradiction since $\|V_1(t)\| = 1$ for all t. Therefore K is a proper subspace and for a generic initial condition $x(0)$, $x(0) - x_*$ will not lie in K and there will exist $\epsilon > 0$ such that

$$\limsup_{t\to\infty} |\langle x(0) - x(*), V_1(t)\rangle| \geq \epsilon.$$

So, by Lemma 1 item 5, we have

$$\limsup_{t\to\infty} \frac{\log \|x[t] - x(*)\|}{t} \geq \limsup_{t\to\infty} \frac{\log \|M_{ii}^{\nu_i(t)}\| + \log \epsilon}{t} = \lim_{t\to\infty} \frac{\nu_i(t)}{t} \log \rho(M_{ii}).$$

The final result follows by taking the maximum over all of the processors, i.e. the maximum over i. $\square$

Theorem 2 shows that the performance of APJ is always held back by its slowest sub-system. In our next result we show that there is a limit to how poorly APJ can perform, which is determined by the length of the shortest path through the DAG.

Recall that for a path σ through the DAG we define the length $L(\sigma)$ to be the number of transitions in σ. The length of the shortest path from an initial condition to a current update is then given by

$$s(t) = \min\{L(\sigma) : \sigma \in \cup_{i,j=1,\ldots,N} P\big(i, j, \nu_i(t)\big)\},$$

and the length of the longest path by

$$l(t) = \max\{L(\sigma) : \sigma \in \cup_{i,j=1,\ldots,N} P\big(i, j, \nu_i(t)\big)\}.$$

The *shortest path growth rate* s^* and *longest path grow rate* l^* are defined by

$$s_* = \lim_{t\to\infty} \frac{s(t)}{t}, \quad l_* = \lim_{t\to\infty} \frac{l(t)}{t}.$$

In our analysis we will assume that there exists $b, B \in \mathbb{R}$ with $0 < t/B \leq s(t) \leq l(t) \leq t/b < \infty$. These bounds could be obtained by first bounding the maximum and minimum length of time between a vector update being used in a computation and it originally being created. Provided this time is bounded above by some $B \in \mathbb{R}$, i.e. provided no communication channel shuts down indefinitely, then we will have $0 < t/B \leq s(t)$. Likewise so long as this time can be bounded below by some $b \in \mathbb{R}$ with $b > 0$, i.e. provided instant communication and computation is not possible, then we will have $l(t) < t/b$. Note that we are not using the exact values of b, B in our bounds, we just need to assume that they exist.

Theorem 3 (Shortest path performance guarantee) *Suppose that the problem $Ax = b$ has non-negative Jacobi iteration matrix M, which we iterate asynchronously with input index Ψ then*

$$\|\Omega(\Psi, M, t)\| \leq \|\sum_{k=s(t)}^{l(t)} M^k\|.$$

If $\rho(M) < 1$ we have

$$\lim_{t\to\infty} \frac{\log \|x[t] - x(*)\|}{t} \le s^* \log[\rho(M)].$$

Otherwise, if $\rho(M) > 1$ then we have

$$\lim_{t\to\infty} \frac{\log \|x[t] - x(*)\|}{t} \le l^* \log[\rho(M)].$$

Proof For a sequence $\varsigma \in \{1, \dots, N\}^{\ell+1}$ let

$$W(\varsigma) = M_{\varsigma(\ell),\varsigma(\ell-1)} \times \cdots \times M_{\varsigma(1),\varsigma(0)}.$$

Now let $S(i,j,\ell) = \{\varsigma \in \{1,\dots,N\}^{\ell+1} : \varsigma(0) = j, \varsigma(\ell) = i\}$ be the set of sequences of length $\ell + 1$ that start with j and end with i. It is easy to prove that the block submatrices of M^k are given by

$$\left[M^k\right]_{ij} = \sum_{\varsigma \in S(i,j,k)} W(\varsigma).$$

and therefore that

$$\left[\sum_{k=s(t)}^{l(t)} M^k\right]_{ij} = \sum_{\varsigma \in \cup_{k=s(t)}^{l(t)} S(i,j,k)} W(\varsigma). \tag{17}$$

From Theorem 1 we have

$$\Omega(\Psi, M, t)_{ij} = \sum_{\sigma \in P\left(i,j,\nu_i(t)\right)} W(\sigma), \tag{18}$$

where this sum is taken over all paths through the DAG starting at $x_j(0)$ and ending at $x_i\big(\nu_i(t)\big)$. Any such path $\sigma \in P\big(i,j,\nu_i(t)\big)$ can be mapped to a unique sequence $\varsigma \in \cup_{k=s(t)}^{l(t)} S(i,j,k)$, with the same weight $W(\varsigma) = W(\sigma)$, by setting

$$\varsigma(m) = \Pi\big(\sigma(m)\big), \quad \text{for } m = 0, \dots, L(\sigma),$$

where $\Pi\big(x_i(k)\big) = i$, for $i = 1,\dots,N$, $k = 1, 2, \dots$ Therefore the terms summed over in (18) form a subset of those summed over in (17), so by non-negativity we have

$$\|\Omega(\Psi, M, t)\| \le \|\sum_{k=s(t)}^{l(t)} M^k\|. \tag{19}$$

For the second result take

$$\lim_{t\to\infty} \frac{\log \|x[t] - x(*)\|}{t} \le \lim_{t\to\infty} \frac{\log \|\Omega(\Psi, M, t)\|}{t} + \frac{\log \|x[0] - x(*)\|}{t}.$$

The second term in the RHS converges to zero and the first term can be bounded using (19)

$$\begin{aligned}\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t} &\leq \lim_{t\to\infty}\frac{\log\|\sum_{k=s(t)}^{l(t)}M^k\|}{t}\\ &\leq \lim_{t\to\infty}\frac{\log\big(l(t)-s(t)+1\big)}{t}+\frac{\log\max_{k=s(t)}^{l(t)}\|M^k\|}{t},\end{aligned}$$

the first term in the RHS converges to zero since $0 < s(t) \leq l(t)$ and since we have assumed that $l(t)$ grows linearly in t.

For $\rho(M) < 1$, from Lemma 1 item 3, we have

$$\begin{aligned}\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t} &\leq \lim_{t\to\infty}\frac{\log\max_{k=s(t)}^{l(t)}\|M^k\|}{t}\\ &\leq \lim_{t\to\infty}\frac{\log\|M^{s(t)}\|}{t}+\frac{\log\max_{k=0}^{l(t)-s(t)}\|M^k\|}{t}.\end{aligned}$$

From Lemma 1 item 6, the second term in the RHS converges to zero. From Lemma 1 item 5 we have

$$\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t}\leq\lim_{t\to\infty}\frac{\log\|M^{s(t)}\|}{t}=\lim_{t\to\infty}\frac{s(t)}{t}\log\big(\rho(M)\big)=s^*\log\big(\rho(M)\big).$$

For $\rho(M) > 1$ we have

$$\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t}\leq\lim_{t\to\infty}\frac{\log\max_{k=s(t)}^{l(t)}\|M^k\|}{t}=\lim_{t\to\infty}\frac{l(t)}{t}\lim_{k\to\infty}\frac{\log\max_{m=1}^{k}\|M^m\|}{k}.$$

From Lemma 1 item 7, we have

$$\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t}\leq l^*\log\big(\rho(M)\big).\ \square$$

Corollary 1 *Suppose that the problem $Ax = b$ has (not necessarily non-negative) Jacobi iteration matrix M with $\rho(|M|) < 1$, where $|M|$ is the componentwise absolute value of M. Take everything else as in the statement of theorem 3. We have the bound*

$$\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t}\leq s^*\log\big(\rho(|M|)\big).$$

Proof This follows by considering the two different error operators $\Omega(\Psi, M, t)$, and $\Omega(\Psi, |M|, t)$. Each is a sum over paths through the same DAG, but the sum in $\Omega(\Psi, |M|, t)$ is taken over the absolute value of the terms in the sum of $\Omega(\Psi, M, t)$, so that

$$\begin{aligned}\lim_{t\to\infty}\frac{\log\|x[t]-x(*)\|}{t} &\leq \lim_{t\to\infty}\frac{\log\|\Omega(\Psi,M,t)\|}{t}\\ &\leq \lim_{t\to\infty}\frac{\log\|\Omega(\Psi,|M|,t)\|}{t}\\ &\leq s^*\log\big(\rho(|M|)\big).\ \square\end{aligned}$$

Corollary 1 guarantees APJ exponential convergence provided no communication channel shuts down indefinitely and provided $\rho(|M|) < 1$. The second condition is guaranteed if the problem matrix A is row-wise diagonal dominant. Of course $\rho(|M|) < 1$ is not a necessary condition for convergence. For example there are many matrices for which $\rho(M) < 1$ but $\rho(|M|) > 1$, in which case the synchronous iteration is stable but the condition is not satisfied. As we will discuss a little more in the next Section, proving general convergence results that are valid for any DAG is extremely difficult.

4 General case

In the general case, when the iteration matrix M is not necessarily real or non-negative, the situation is considerably more difficult. Theorem 2 (weakest link performance barrier) does not apply and Theorem 3 (shortest path performance guarantee) will only prove convergence if $\rho(|M|) < 1$.

In this section we will give simple examples of iteration matrices that are stable under synchronous iteration but unstable under asynchronous iteration and vice versa. Of course the stability of an asynchronous iteration will depend strongly on the precise properties of the DAG so that our example of a matrix which is stable under synchronous iteration but unstable under asynchronous iteration can not possibly be unstable for any DAG, as the synchronous iteration itself can be represented with a DAG. Instead we will give a simple random model for generating an asynchronous DAG and show that with overwhelming probability the resulting asynchronous iteration will have the prescribed stability properties.

We do this by examining the Lyapunov exponents of the asynchronous systems. The theory for Lyapuov exponents is mostly focused towards analyzing the stability properties of invariant sets to non-linear dynamical systems and as such the important results are usually stated in these terms, for example see the text book [17]. However the problem of determining the Lyapunov exponent of a set of matrices has received some attention, see for example the papers [10,11,16] and the references therein.

Let $\{M(1), \ldots, M(N)\} \subset C^{n \times n}$ be a finite set of matrices and let $\alpha \in \{1, \ldots, N\}^{\mathbb{N}}$ be a random variable such that $\alpha_1, \alpha_2, \ldots$ are independent and identically distributed with $\mathbb{P}\{\alpha_k = m\} = 1/N$ for all $k \in \mathbb{N}$, $m \in \{1, \ldots, N\}$. The multiplicative ergodic Theorem ([10], Theorem 2) states that there exists $\lambda \in \mathbb{R}$ such that for all $\epsilon > 0$

$$\lim_{k \to \infty} \mathbb{P}\{|\frac{\log \|M(\alpha_k) \times \cdots \times M(\alpha_1)\|}{k} - \lambda| \leq \epsilon\} = 1. \tag{20}$$

We call λ the *Lyapunov exponent.* The multiplicative ergodic theorem tells us that, with overwhelming probability, the growth rate of the norm of the product that we observe from a sufficiently long randomly generated sequence will converge to the Lyapunov exponent. We say that the following holds almost surely, since it is true for a measure one set of sequences

$$\lim_{k\to\infty} \frac{\log \|M(\alpha_k) \times \cdots \times M(\alpha_1)\|}{k} = \lambda. \tag{21}$$

To approximate the spectral radius numerically we evaluate the above quantity for a large but finite value of k, obtaining the sequence $\alpha_1, \ldots, \alpha_k$ from a random number generator.

A related quantity is the *joint spectral radius*, which is defined by

$$\rho = \lim_{k\to\infty} \max_{\alpha\in\{1,\ldots,N\}^k} \|M(\alpha_k) \times \cdots \times M(\alpha_1)\|^{1/k}. \tag{22}$$

The multiplicative ergodic theorem tells us that for almost all sequences α we will observe the same growth rate of the product. The spectral radius instead tells us about the most extreme growth rate that is possible. We always have $\log(\rho) \geq \lambda$ and typically we have $\log(\rho) > \lambda$. In examples where $\log(\rho) > \lambda$, the set of sequences whose growth rates attain the joint spectral radius form a set of measure zero. See the special issue journal on the subject [4].

We can think of the product of a random sequence of matrices as an asynchronous iteration. For $M \in \mathbb{C}^{n\times n}$ define the set of matrices $\{M(1), \ldots, M(N)\}$ by their block structure

$$M(m)_{ij} = \begin{cases} M_{ij} & \text{if } i = m, \\ I & \text{if } i = j \neq m, \\ 0 & \text{otherwise.} \end{cases}$$

Now suppose that we have N processors that each update with independent, mean-1, exponentially distributed inter-update times. The sequence $\alpha \in \{1, \ldots, N\}^{\mathbb{N}}$ in which the processors update then has the distribution described previously. Suppose further that whenever a processor updates it uses the most recently computed values from its neighbors' states. With these assumptions the DAG is completely determined by the update sequence α and the error operator is given by

$$\Omega(\Psi, M, t) = M(\alpha_{\nu(t)}) \times \cdots \times M(\alpha_1), \tag{23}$$

where $\nu(t)$ is the total number of processor updates to have occurred by time t. The rate of convergence of this asynchronous iteration is given by

$$-\lim_{t\to\infty} \frac{\log \|\Omega(\Psi, M, t)\|}{t} = -\lim_{t\to\infty} \frac{\nu(t)\lambda}{t} = -N\lambda$$

where λ is the Lyapunov exponent of $\{M(1), \ldots, M(N)\}$.

The fair synchronous comparison with this rate of convergence is given by $-\log\big(\rho(M)\big)$, since this assumes an average update rate of 1 for the synchronous iteration.

Example 2 For $\theta \in \mathbb{R}_+$ let

$$M = \begin{bmatrix} -\theta & \theta \\ \theta & -\theta \end{bmatrix}, \quad M(1) = \begin{bmatrix} -\theta & \theta \\ 0 & 1 \end{bmatrix}, \quad M(2) = \begin{bmatrix} 1 & 0 \\ \theta & -\theta \end{bmatrix}.$$

The rate of convergence of the synchronous iteration is given by

$$-\log \rho(M) = -\log(2) - \log(\theta),$$

so that the synchronous iteration is stable for $\theta < 1/2$ and unstable for $\theta > 1/2$.

We compute the asynchronous rate of convergence for different values of θ via Monte Carlo simulation. To do this we randomly sample the entries in α and form a long product (23), which we repeatedly normalize, keeping track of the sum of the logs of the normalizing factors, which we use to compute the rate of convergence. See Figure 2 (a). We also plot the shortest path lower bound from Corollary 1. For this bound we need the growth rates of the the shortest/longest paths through the DAG. By considering the modeling assumptions we calculate these exactly to be $s_* = 2/3$ and $l_* = 2$.

The numerical experiments show that the asynchronous iteration is stable for $\theta < 1$ and unstable for $\theta > 1$. It is possible to prove that the asynchronous iteration does not diverge for $\theta = 1$, however, instead of converging to zero the sequence of matrix norms behaves like a recurrent random walk. Here we will only prove that the asynchronous iteration is stable for $\theta = 1/\sqrt{2}$, which provides an examples of a matrix which is unstable under synchronous iteration but stable under asynchronous iteration.

There are 8 different products of $M(1)$ and $M(2)$ of length 3: $P(1) = M(1)M(1)M(1)$, $P(2) = M(1)M(1)M(2)$, ... , $P(8) = M(2)M(2)M(2)$. For any sequence $\alpha \in \{1,2\}^{\mathbb{N}}$ there exists a sequence $\beta \in \{1,\dots,8\}^{\mathbb{N}}$ such that

$$M(\alpha_{3k}) \times \cdots \times M(\alpha_1) = P(\beta_k) \times \cdots \times P(\beta_1), \quad \text{for all } k \geq 0.$$

Using Lemma 1 item 3 we have

$$\|M(\alpha_{3k}) \times \cdots \times M(\alpha_1)\| \leq \prod_{m=1}^{k} \|P(\beta_m)\| \tag{24}$$

$$= \prod_{\ell=1}^{8} \|P(\ell)\|^{\#(\ell,\beta,k)}, \tag{25}$$

where $\#(\ell,\beta,k)$ is the number of times that the symbol ℓ appears in the first k terms of the sequence β. Since each possible term in α appears with equal probability, the different possible terms in β also appear with equal probability and for all $\epsilon > 0$

$$\lim_{k\to\infty} \mathbb{P}\{|\frac{\#(\ell,\beta,k)}{k} - \frac{1}{8}| \leq \epsilon\} = 1.$$

Applying the bound (24) to equation (21) gives, for almost all sequences $\beta \in \{1,\dots,8\}^{\mathbb{N}}$

$$\lambda \leq \lim_{k\to\infty} \frac{\log\left(\prod_{\ell=1}^{8} \|P(\ell)\|^{\#(\ell,\beta,k)}\right)}{k} \tag{26}$$

$$= \sum_{\ell=1}^{8} \log\|P(\ell)\| \lim_{k\to\infty} \frac{\#(\ell,\beta,k)}{k} \tag{27}$$

$$= \sum_{\ell=1}^{8} \log\|P(\ell)\| \frac{1}{8}. \tag{28}$$

This bound guarantees that the asynchronous iteration is stable whenever $r < 1$, where

$$r = \prod_{\ell=1}^{8} \|P(\ell)\|.$$

Using Wolfram Mathematica we calculated the quantity r exactly for $\theta = 1/\sqrt{2}$

$$r = 1/64\left(10 - 3\sqrt{2} + \sqrt{110 - 60\sqrt{2}}\right)\left(10 - 5\sqrt{2} + \sqrt{2(71 - 50\sqrt{2})}\right).$$

Using elementary techniques it is simple to prove that this expression is less than 1. The numerical value is approximately 0.621.

Example 3 For $\theta \in \mathbb{R}_+$ let

$$M = \begin{bmatrix} \theta & -\theta \\ \theta & -\theta \end{bmatrix}, \quad M(1) = \begin{bmatrix} \theta & -\theta \\ 0 & 1 \end{bmatrix}, \quad M(2) = \begin{bmatrix} 1 & 0 \\ \theta & -\theta \end{bmatrix}.$$

The rate of convergence of the synchronous iteration is given by

$$-\log\rho(M) = \infty,$$

so that the synchronous iteration is stable for all values of $\theta \in \mathbb{R}$. Since $M^2 = 0$ so the synchronous iteration always converges after two updates.

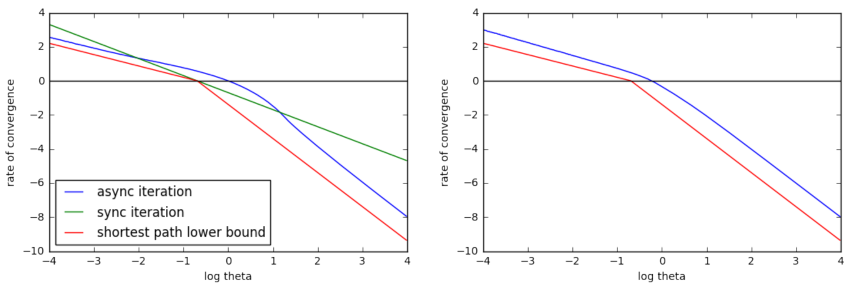
The asynchronous iteration will not converge in finite time for any initial condition since both of the asynchronous iteration matrices are non-singular. See Figure 2 (b). Whilst we cannot detect the exact value of θ at which the asynchronous iteration switches from being stable to being unstable, we can easily prove that for $\theta > 1$ the asynchronous iteration is unstable, which provides a range of examples of matrices which are stable under synchronous iteration but unstable under asynchronous iteration.

For any 2×2 matrix M we have

$$|\det(M)| \leq \rho(M)^2 \leq \|M\|^2,$$

The determinants of the asynchronous iteration matrices satisfy

$$|\det\left(M(1)\right)| = |\det\left(M(2)\right)| = \theta.$$



(a) Example 2 (b) Example 3

Fig. 2

So that for $\theta > 1$ we have, for almost all sequences α

$$\begin{aligned}\lambda &= \lim_{n\to\infty} \frac{\log \|M(\alpha_n)\times\cdots\times M(\alpha_1)\|}{n}\\ &\geq \lim_{n\to\infty} \frac{\log \rho\big(M(\alpha_n)\times\cdots\times M(\alpha_1)\big)}{n}\\ &\geq \lim_{n\to\infty} \frac{\log \det\big(M(\alpha_n)\times\cdots\times M(\alpha_1)\big)}{2n}\\ &= \frac{1}{2}\log\theta > 0.\end{aligned}$$

5 Numerical Experiments

In this section we report the results of our numerical experiments. We randomly generate a number of different test problems which we then solve using both synchronous and asynchronous Jacobi's method. For each Jacobi solve we make a precise recording of the DAG associated with the computation. We are then able to compare the performance of the synchronous and asynchronous algorithms to various statistical properties of the DAGs including the bounds that we proved in Section 3.

Our numerical experiments were carried out using the Balena High Performance Computing facility at the University of Bath. Balena has 3,072 general purpose Intel "IvyBridge" cores and over 18 TiB of main memory. The entire system is connected by Intel TrueScale Infiniband at 40Gb/sec. When solving the linear system $Ax = b$ using synchronous or asynchronous Jacobi's method the iteration matrix M and vectors x and c are partitioned such that each processor stores m rows of M and the corresponding m entries of x and c. When processor i forms the matrix-vector product in (1), some of the vector elements required will be stored locally on processor i and some will be stored (and updated) on other remote processors; these latter elements are

Table 2: matrix parameteres

Matrix ID Letter	d_1	d_2
a	1	1
b	3	2
c	5	5
d	10	10

communicated during the update phase. In constructing the problems for our experiments on 2 computational nodes, we generate the problem matrix A in such a way that for each given processor, d_1 of the remote processors will be found on the same computational node and d_2 will be on the other node.

We generate a range of test problems as follows. The problem matrix A is defined in terms of its 32×32 submatrices $(A_{ij} \in \mathbb{C}^{m\times m})_{i,j=1}^{32}$, where $m = 100,000$ is the number of rows per processor. We define the interprocessor connectivity matrix $C \in \{0,1\}^{32\times 32}$ by

$$C = \begin{bmatrix} M_{d_1+1} & M_{d_2} \\ M_{d_2} & M_{d_1+1} \end{bmatrix},$$

where M_d is the 16×16 matrix with entries given by

$$[M_d]_{ij} = \begin{cases} 1 \text{ if } 0 \le j - i < d \pmod{16} \\ 0 \qquad\qquad \text{otherwise.} \end{cases}$$

The block submatrix A_{ij} is non-zero, and hence processor i receives input from processor j, if and only if $c_{ij} = 1$. We call $d_1 + d_2$ the number of *processor neighbors*.

Each row of A is given exactly 10 non-zero, off-diagonal entries. Five of these entries are equal to $v_1 = 1$ and are placed uniformly at random inside the diagonal block. The remaining five entries are equal to v_2, which is varied between test problems, and are placed uniformly at random in the off diagonal blocks that are allowed by the interprocessor connectivity matrix. We call v_2 the *coupling strength.* The diagonal entries of A are all set to $v_d = -\delta(5+5v_2)$, where $\delta = 1.005$ determines the degree of diagonal dominance.

We generate a total of 28 different test problems over a range of parameters. See Tables 2 and 3. The letter part of the matrix ID refers to the number of processor neighbors and the number part refers to the coupling strength. Table 3 also records the spectral radius of the iteration matrix, $\rho(M)$ and the spectral radius of a diagonal block submatrix $\rho(M_{ii})$, note that this is the same for all $i = 1, \dots, N$, and that both spectral radii are independent of the matrix ID letter. Observe that for the weakly coupled problems, with smaller values of v_2, the spectral radius of the diagonal block submatrix is very close to that of the full matrix.

For each test matrix A we solve the system $Ax = b$ using both synchronous and asynchronous parallel Jacobi's method. We set $b = Ax(*)$ where

$$x_i(*) = \sin(2\pi i/m), \quad \text{for } i = 1, \dots, n, \tag{29}$$

Table 3: matrix parameteres and spectra

Matrix ID Number	v_2	v_d	$\rho(M)$	$\rho(M_{ii})$	$\log(\rho(M))/\log(\rho(M_{ii}))$
1	10	-55.27	0.9950	0.0904	0.0020
2	1	-10.04	0.9950	0.4975	0.0071
3	0.1	-5.527	0.9950	0.9045	0.0497
4	0.01	-55.27	0.9950	0.9851	0.3338
5	0.001	-5.025	0.9950	0.9940	0.8330
6	1e-04	-5.025	0.9950	0.9949	0.9803
7	1e-05	-5.025	0.9950	0.9950	0.9979

and use the initial guess

$$x_i(0) = 1, \quad \text{for } i = 1, \ldots, n. \tag{30}$$

We also use the termination conditions described in Section 2.2 with $\epsilon =$ 1e$-$12.

We instrument the algorithms to record the DAG during the course of the computation. When a processor i broadcasts its kth update it adds a header to that data including the time at which it was produced as well as the iteration number k. Then if processor j receives this data and uses it in its ℓth iteration, say, processor j will record a log that it used processor i's kth update on its ℓth iteration as well as recoding the time that that data was created by processor i and the time that its own computation was completed. At the end of the computation this data can be aggregated to completely reconstruct the DAG of the computation.

5.1 Synchronous iteration

For the synchronous variant we record the total number of iterations needed for convergence, the final residual, the final error and the total time taken in seconds. We also calculate the update rate in updates per second and the observed rate of convergence which is calculated by

$$\text{obs' rate of conv'} = \frac{\log\left(\frac{\text{2-norm of error in initial condition}}{\text{2-norm of final error}}\right)}{\text{time taken}},$$

where the initial error is given by $\sqrt{\sum_{i=1}^{n}\left(\sin(2\pi i/m) - 1\right)^2} = 2190$. See Table 4.

5.2 Asynchronous iteration

During asynchronous iteration some processors will update more than others. Figure 3 (a) shows the update rates of all of the different processors recorded during the asynchronous iteration on problem C1. Note that all of the asynchronous processors update faster than the synchronous rate, but that there

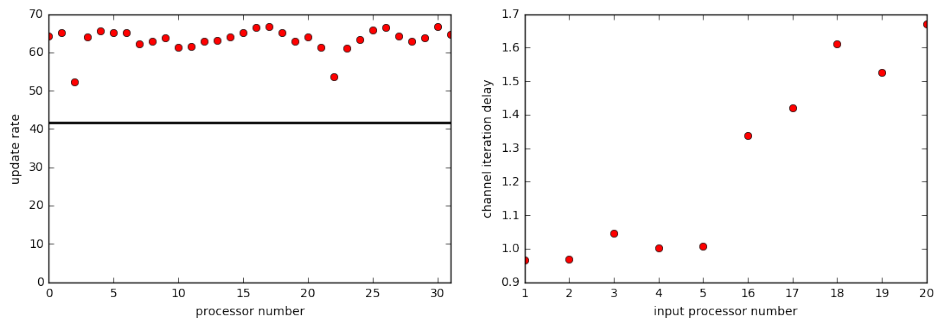
Table 4: sync performance

Matrix ID	its'	res' (log 10)	error (log 10)	time	up' rate	obs' rate of conv'
a1	2994	-3.793	-3.232	53.70	55.74	0.1223
a2	3113	-4.791	-3.490	57.14	54.47	0.1195
a3	3432	-5.742	-4.181	61.80	55.53	0.1217
a4	3595	-6.132	-4.534	65.63	54.76	0.1199
a5	3608	-6.164	-4.562	64.42	56.00	0.1226
a6	3708	-6.333	-4.733	67.27	55.11	0.1200
a7	3737	-6.361	-4.759	66.96	55.80	0.1209
b1	3013	-3.834	-3.273	63.76	47.25	0.1037
b2	3132	-4.832	-3.531	66.63	47.00	0.1031
b3	3447	-5.774	-4.213	73.06	47.17	0.1033
b4	3596	-6.134	-4.536	75.38	47.70	0.1044
b5	3612	-6.172	-4.571	75.78	47.66	0.1043
b6	3713	-6.338	-4.738	79.47	46.72	0.1016
b7	3737	-6.360	-4.758	79.54	46.98	0.1018
c1	2993	-3.791	-3.230	71.69	41.74	0.0916
c2	3112	-4.789	-3.488	74.92	41.53	0.0911
c3	3430	-5.737	-4.177	82.74	41.45	0.0908
c4	3595	-6.132	-4.534	85.91	41.84	0.0916
c5	3607	-6.161	-4.560	87.56	41.19	0.0902
c6	3707	-6.331	-4.731	89.52	41.40	0.0901
c7	3737	-6.361	-4.759	88.59	42.17	0.0914
d1	2994	-3.793	-3.232	75.44	39.68	0.0871
d2	3113	-4.791	-3.490	79.04	39.38	0.0864
d3	3431	-5.739	-4.179	86.72	39.56	0.0867
d4	3595	-6.132	-4.534	91.32	39.36	0.0862
d5	3608	-6.163	-4.562	91.94	39.24	0.0859
d6	3708	-6.333	-4.733	96.44	38.44	0.0837
d7	3737	-6.361	-4.759	95.34	39.19	0.0849

is quite a lot of variation and two processors, one from each core, have significantly slower rates than the rest of the group. This distribution of update rates is characteristic of all of the other test problems.

For the asynchronous iteration we record the mean number of individual processor iterations needed for convergence, the final residual, the final error and the total time taken in seconds. We calculate the mean update rate in updates per second and the observed rate of convergence.

We also record the mean iteration delay, which is defined as follows. If processor i uses processor j's kth update on its ℓth iteration and at the moment that that computation is finished processor j has performed $k+m$ iterations, then the iteration delay of the communication is equal to m. In the synchronous iteration every communication has iteration delay exactly equal to one. Figure 3 (b) shows the mean iteration delay of communications to processor number 0 during the asynchronous iteration on problem C1. Communications from processors on the same node have mean iteration delay roughly equal to one but delays from processors on the neighboring node are larger. For each problem we record the average iteration delay over all channels, the



(a) Asynchronous processor update rates, synchronous update rate marked with horizontal line.

(b) Mean iteration delay of inputs to processor 0.

Fig. 3: Processor update rates and mean iteration delay for inputs to processor 0 for for problem C1.

maximum iteration delay over all channels and the maximum iteration delay over all channels for the final ten iterations. See Table 8.

5.3 Comparison

To make a direct comparison between the performance of the synchronous and asynchronous solvers we record the following quantities. *Boost* measures the increase in mean update rate obtained by switching from synchronous to asynchronous iteration. *Effectiveness* is the average rate of convergence per processor update. It can be calculated by

$$\text{effectiveness} = \frac{\text{rate of convergence}}{\text{update rate}}.$$

We also record the effectiveness ratio which is the asynchronous effectiveness divided by the synchronous effectiveness. *Speedup* measures the improvement in rate of convergence obtained by switching from synchronous to asynchronous iteration. See Table 6 and Figure 4 (a,b,c).

For the synchronous iteration the update rate depends only on the number of processor neighbors and not on the coupling strength. The synchronous effectiveness is constant for all of the tests and is approximately equal to $\log\big(\rho(M)\big) = 2.166\text{e}{-3}$. The rate of convergence of the synchronous iteration is therefore determined wholly by the update rate, which decreases with the number of processor connections.

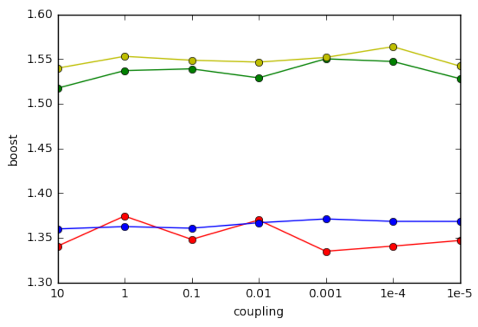
For the asynchronous iteration the update rate also depends mainly on the number of processor neighbors, although there is slightly more variance between runs with the same number of neighbors. A synchronous system with

Table 5: async performance

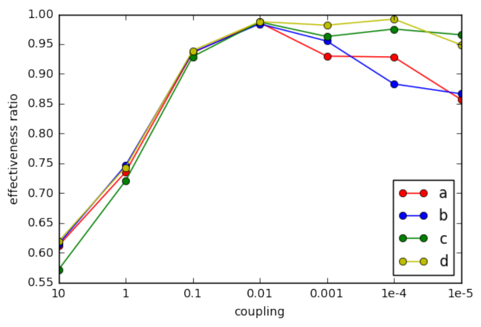
Matrix ID	mean its'	res' (log 10)	error (log 10)	time	mean up' rate	obs' rate of conv'
a1	5089	-3.793	-3.489	68.10	74.72	0.1002
a2	4359	-4.791	-3.689	58.23	74.85	0.1207
a3	3719	-5.742	-4.283	49.67	74.86	0.1534
a4	3716	-6.132	-4.686	49.53	75.01	0.1620
a5	4186	-6.164	-5.183	55.99	74.75	0.1522
a6	4273	-6.333	-5.295	57.83	73.88	0.1493
a7	4660	-6.361	-5.316	62.00	75.15	0.1396
b1	5161	-3.834	-3.617	80.32	64.24	0.0866
b2	4340	-4.832	-3.765	67.77	64.03	0.1048
b3	3758	-5.774	-4.371	58.55	64.18	0.1317
b4	3772	-6.134	-4.788	57.85	65.19	0.1405
b5	4065	-6.172	-5.163	62.21	65.34	0.1366
b6	4501	-6.338	-5.307	70.40	63.92	0.1228
b7	4604	-6.360	-5.305	71.62	64.27	0.1207
c1	5557	-3.791	-3.626	87.72	63.34	0.0794
c2	4462	-4.789	-3.712	69.89	63.83	0.1009
c3	3777	-5.737	-4.348	59.20	63.79	0.1298
c4	3716	-6.132	-4.693	58.08	63.97	0.1383
c5	4013	-6.161	-5.121	62.84	63.85	0.1346
c6	4039	-6.331	-5.236	63.04	64.06	0.1360
c7	4087	-6.361	-5.213	63.41	64.44	0.1348
d1	5132	-3.793	-3.629	83.98	61.10	0.0829
d2	4351	-4.791	-3.753	71.14	61.16	0.0997
d3	3724	-5.739	-4.323	60.78	61.26	0.1260
d4	3715	-6.132	-4.696	61.02	60.87	0.1316
d5	3859	-6.163	-4.956	63.37	60.89	0.1309
d6	3894	-6.333	-5.070	64.77	60.11	0.1298
d7	4205	-6.361	-5.307	69.57	60.43	0.1242

more inter-processor connections will naturally spend more idle time waiting for fresh inputs. Thus we expect that boost increases with the number of processor neighbors. Figure 4 (a) shows that the problems divide into two groups. Problems with 2 or 5 processor neighbors (ID letters a and b) have a boost of around 1.35 and problems with 10 or 20 processor neighbors (ID letters c and d) have a boost of around 1.55.

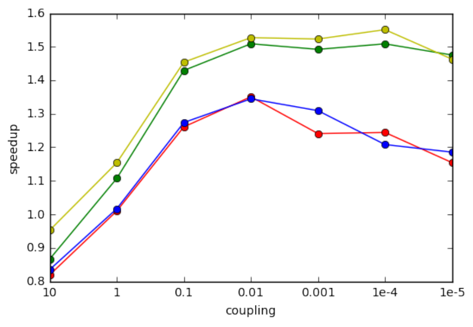
Unlike synchronous effectiveness, asynchronous effectiveness is not constant and varies with coupling strength as well as the number of processor neighbors. Since the synchronous effectiveness is constant the asynchronous effectiveness and effectiveness ratio are directly proportional. Processors in the asynchronous iteration can use out-of-date data from their neighbors. So we expect that the asynchronous iteration will be less effective for strongly coupled problems, where the communicated data has a greater weight. Figure 4 (b) shows that the effectiveness ratio is smallest for strongly coupled problems and increases to nearly 1 at a coupling strength of 0.01. For weakly coupled problems the effectiveness ratio drops off slightly, more so for the systems with fewer processor neighbors. In Section 5.4 we explain this dip using the weakest link performance barrier (Theorem 2).



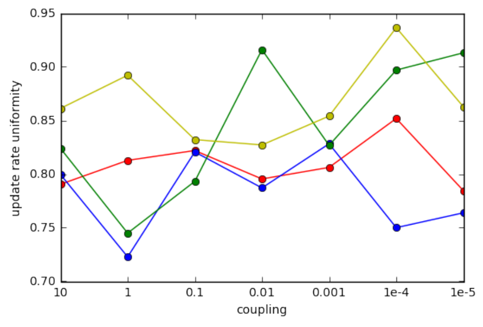
(a) Boost



(b) Effectiveness ratio



(c) Speedup



(d) Update rate uniformity

Fig. 4

Speedup is equal to boost times the effectiveness ratio. Figure 4 (c) shows that speedup follows a similar pattern to the effectiveness ratio, except that the larger boost value for problems with 10 or 20 processor neighbors (ID letters c and d) means that these problems also have a larger speedup. All problems with a coupling strength of 1 or less have a speedup of greater than one, meaning that the asynchronous iteration converges faster.

5.4 Weakest link upper bound

Theorem 2 gives an upper bound on the asymptotic rate of convergence of the asynchronous iteration. This bound can be calculated by

$$\text{asy' rate of conv'} \leq -\big(\text{update rate of slowest processor}\big) \times \log\big(\rho(M_{ii})\big).$$

See Table 7. Note that there are several problems for which the weakest link upper bound (WLUB) is less than the observed rate of convergence. This is due to the fact that we are measuring the rate of convergence over a finite

Table 6: performance comparison

		(e-3)			
Matrix ID	boost	sync' effect'	async' effect'	effect' ratio	speedup
a1	1.340	2.195	1.342	0.611	0.819
a2	1.374	2.194	1.612	0.734	1.009
a3	1.348	2.191	2.050	0.935	1.260
a4	1.369	2.190	2.160	0.986	1.350
a5	1.334	2.190	2.036	0.929	1.240
a6	1.340	2.177	2.021	0.928	1.244
a7	1.346	2.167	1.857	0.857	1.154
b1	1.359	2.195	1.348	0.614	0.835
b2	1.362	2.194	1.637	0.746	1.016
b3	1.360	2.191	2.052	0.936	1.273
b4	1.366	2.190	2.155	0.983	1.344
b5	1.370	2.190	2.091	0.955	1.309
b6	1.368	2.175	1.921	0.883	1.208
b7	1.368	2.167	1.877	0.866	1.185
c1	1.517	2.195	1.253	0.571	0.866
c2	1.536	2.194	1.580	0.720	1.107
c3	1.538	2.191	2.035	0.928	1.429
c4	1.528	2.190	2.162	0.987	1.508
c5	1.550	2.190	2.108	0.962	1.492
c6	1.547	2.177	2.123	0.975	1.508
c7	1.527	2.167	2.092	0.965	1.475
d1	1.539	2.195	1.358	0.618	0.952
d2	1.552	2.194	1.630	0.742	1.153
d3	1.548	2.191	2.058	0.939	1.454
d4	1.546	2.190	2.163	0.987	1.527
d5	1.551	2.190	2.150	0.981	1.523
d6	1.563	2.177	2.159	0.991	1.551
d7	1.542	2.167	2.056	0.948	1.463

length of time and as such we only have an approximate upper bound. If we take the first part of Theorem 2, equation (16), take logs and divide by time we obtain

$$\frac{-\log \|\Omega(\Psi, M, t)\|}{t} \leq -\big(\text{update rate of slowest processor}\big) \times \log\big(\rho(M_{ii})\big).$$

where $\|\Omega(\Psi, M, t)\|$ is the operator 2-norm of the error operator. The quantity on the LHS is therefore the smallest possible rate of convergence that we could observe from any initial condition. Just like in a synchronous iteration, some modes of error decay much faster than others during asynchronous iteration and a typical initial condition will include a combination of different error modes. The fast decaying modes decay in a small number of iterations and the slow decaying modes determine the asymptotic rate of convergence. The mismatch between the asymptotic rate of convergence and the observed rate comes from the presence of fast decaying modes in the initial error. One heuristic way to improve the weakest link approximate upper bound is to assume that the slow decaying mode of the asynchronous iteration is given by the Perron eigenvector of the diagonal block associated with the slowest updating processor and that this slow mode dominates the observed rate of convergence. This yields the approximation

$$\|x(t) - x(*)\| \approx \big|\langle u, x(0) - x(*)\rangle\big| \exp\big(-t \times \text{asy' rate of conv'}\big), \tag{31}$$

where $u \in \mathbb{R}^n$ is a vector with block form $u = (u_i \in \mathbb{R}^m)_{i=1}^{32}$. For the slowest updating processor i we have that u_i is the Perron eigenvector of M_{ii}, normalized so that $\|u_i\| = 1$, and otherwise we have $u_i = 0$. We therefore have

$$\text{obs' rate of conv'} \approx \text{asy' rate of conv'} + \frac{\log\left(\frac{\|x(0)-x(*)\|}{\left|\langle u, x(0)-x(*)\rangle\right|}\right)}{\text{time taken}}. \tag{32}$$

Now using the fact that the Perron eigenvector of M_{ii} is a flat vector (with all entries equal) and substituting in (29) and (30), we obtain

$$\frac{\|x(0) - x(*)\|}{\big|\langle u, x(0) - x(*)\rangle\big|} = 6.928.$$

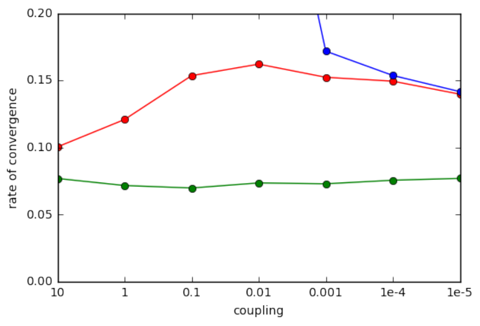
This gives the relationship between observed and asymptotic rates of convergence, which we use to compute the second weakest link approximate upper bound given in Table 7. This adjusted bound (WLUB II) is still less than the observed rate of convergence for two problems, but it is only a small overshoot compared to the unadjusted bound.

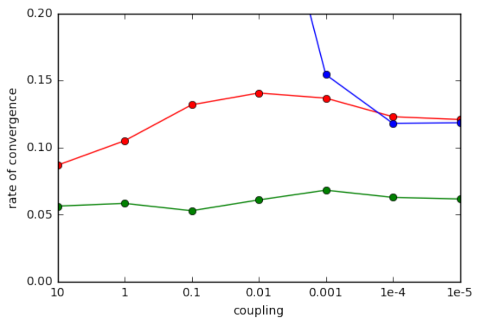
Figure 5 (a,b,c,d) shows that the adjusted weakest link upper bounds give a good approximation to the rate of convergence for weakly coupled problems. For strongly coupled problems the factor $-\log\big(\rho(M_{ii})\big)$ grows very quickly and the upper bounds blow up. This can also be seen in Table 3.

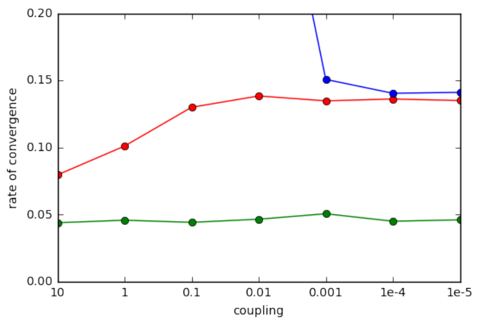
The dip in speedup shown in Figure 4 (c) can now be explained. For very weakly coupled problems the processor iterations are effectively independent from one another and the convergence to solution is held up by the slowest processor(s), which as shown in Figure 3 (a) typically update quite a lot slower than the average rate. For larger values of coupling strength work is effectively shared between the processors and the much smaller local eigenvalue $\rho(M_{ii})$, means that one or two slow processors will not harm the overall rate of convergence. We define the *update rate uniformity* of an asynchronous iteration to be the update rate of the slowest processor divided by the mean update rate. This is plotted in Figure 4 (d). That the problems with fewer processor neighbors (ID letters a and b) have a bigger dip in effectiveness for weak coupling follows from the fact that these problems also have lower values of update rate uniformity. Variation in update rate uniformity has less effect on the asynchronous effectiveness for more strongly coupled problems because for these problems the update rate of the slowest processor is less important.

5.5 Shortest path lower bounds

Theorem 3 gives a lower bound on the rate of convergence. Like the weakest link bound the shortest path bound is a bound on the asymptotic rate of convergence. But since the observed rate of convergence is always greater than the


(a) ID letter a


(b) ID letter b


(c) ID letter c

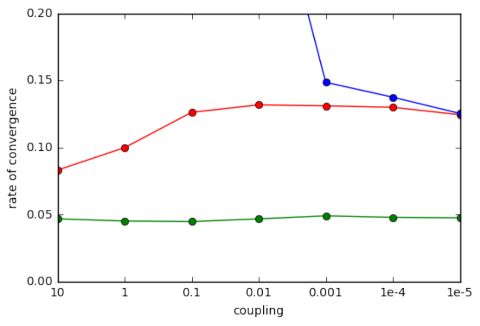

(d) ID letter d

Fig. 5

asymptotic rate, the shortest path bound on the observed rate of convergence is still valid. We should not use heuristic (32) here as it is only valid for weakly coupled problems and we want to apply the shortest path lower bound to all of our problems including the strongly coupled ones.

To compute the length of the shortest path through the DAG we assign each processor update a depth. Initial conditions are given a depth of zero. We then set the depth of subsequent updates to be one plus the minimum of the depths of their inputs. Thus the depth of an update is equal to the length of the shortest path through the DAG from an initial condition to that update. The shortest path length of the DAG is then equal to the minimum depth of a final update. The shortest path growth rate is equal to the shortest path length divided by the time taken to converge and the shortest path lower bound (SPLB) is given by

$$\text{asy' rate of conv'} \geq -\big(\text{shortest path growth rate}\big) \times \log\big(\rho(M)\big).$$

Figure 5 (a,b,c,d) shows that the shortest path bound gives a good estimate for the rate of convergence for strongly coupled systems.

Table 7: weakest link bounds

Matrix ID	min its'	min up' rate	obs' rate of conv'	WLAUB	WLAUB II
a1	4023	59.07	0.100	61.64	61.65
a2	3543	60.84	0.120	18.44	18.46
a3	3057	61.53	0.153	2.680	2.697
a4	2956	59.67	0.162	0.387	0.404
a5	3375	60.27	0.152	0.156	0.171
a6	3640	62.93	0.149	0.139	0.153
a7	3656	58.96	0.139	0.127	0.141
b1	4127	51.37	0.086	53.61	53.62
b2	3137	46.28	0.104	14.03	14.04
b3	3084	52.67	0.131	2.294	2.308
b4	2969	51.31	0.140	0.332	0.347
b5	3367	54.12	0.136	0.140	0.154
b6	3376	47.94	0.122	0.105	0.117
b7	3517	49.10	0.120	0.106	0.118
c1	4579	52.19	0.079	54.47	54.48
c2	3324	47.55	0.100	14.41	14.43
c3	2996	50.60	0.129	2.204	2.218
c4	3404	58.59	0.138	0.380	0.394
c5	3318	52.79	0.134	0.137	0.150
c6	3623	57.46	0.136	0.126	0.140
c7	3732	58.85	0.134	0.127	0.140
d1	4420	52.62	0.082	54.91	54.92
d2	3882	54.56	0.099	16.54	16.55
d3	3099	50.98	0.126	2.220	2.234
d4	3073	50.35	0.131	0.326	0.340
d5	3296	52.00	0.130	0.135	0.148
d6	3647	56.30	0.129	0.124	0.137
d7	3628	52.14	0.124	0.113	0.125
'					

5.6 Error analysis

The error bounds we presented in Section 2.3 are in terms of the matrix condition number $\kappa(A)$, which we cannot know exactly for our randomly generated problem matrices. One way to estimate this quantity is to take the ratio of the forwards error to the residual error. This results in consistent estimates between the synchronous and asynchronous approximate solutions. However, since it is a little circular applying our error bounds using such a condition number estimate, we instead examine the residual error. The synchronous error bound (10) becomes

$$\text{sync' residual error} \leq \epsilon\sqrt{n}\|b\|,$$

and the asynchronous error bound (15) becomes

$$\text{async' residual error} \leq 2\epsilon k_{\max}\sqrt{n}\|b\|.$$

Recall that we use the threshold $\epsilon = 1\text{e}{-12}$. We use the final iteration delay data from Table 8 for $k_{\max}$. Table 9 shows that the residual errors in the

Table 8: short path lower bounds

	iteration delay			shortest path			
Matrix ID	mean	max	final	length	rate	obs' rate of conv'	SPLB
a1	1.168	24	3	2413	35.43	0.100	0.076
a2	1.180	11	3	1922	33.00	0.120	0.071
a3	1.178	5	3	1597	32.14	0.153	0.069
a4	1.172	15	3	1679	33.89	0.162	0.073
a5	1.176	13	3	1881	33.59	0.152	0.072
a6	1.151	8	3	2014	34.82	0.149	0.075
a7	1.181	5	3	2199	35.46	0.139	0.076
b1	1.134	17	3	2081	25.90	0.086	0.056
b2	1.114	5	3	1820	26.85	0.104	0.058
b3	1.132	5	3	1424	24.32	0.131	0.052
b4	1.129	10	4	1622	28.03	0.140	0.060
b5	1.116	8	3	1953	31.39	0.136	0.067
b6	1.114	4	3	2036	28.91	0.122	0.062
b7	1.133	5	4	2032	28.36	0.120	0.061
c1	1.276	7	4	1770	20.17	0.079	0.043
c2	1.228	8	4	1472	21.06	0.100	0.045
c3	1.270	9	4	1203	20.31	0.129	0.044
c4	1.267	8	4	1243	21.39	0.138	0.046
c5	1.242	7	4	1463	23.27	0.134	0.050
c6	1.301	7	4	1305	20.69	0.136	0.044
c7	1.282	9	4	1343	21.17	0.134	0.045
d1	1.132	6	4	1809	21.53	0.082	0.046
d2	1.145	6	5	1478	20.77	0.099	0.045
d3	1.147	5	3	1251	20.58	0.126	0.044
d4	1.124	5	4	1313	21.51	0.131	0.046
d5	1.110	6	4	1430	22.56	0.130	0.048
d6	1.122	6	4	1427	22.03	0.129	0.047
d7	1.144	8	4	1522	21.87	0.124	0.047

asynchronous solution are consistently slightly smaller than in the synchronous case. We believe that this is due to additional iterations being performed during the delay between each processor reaching its local stopping condition and the termination command being generated and reaching them.

Conclusion

We introduced a formalism for describing the behavior of APJ in terms of its DAG. Using this approach we proved error bounds and showed how the rate of convergence can be linked to statistical properties of the DAG, namely the update rate of the slowest processor and the shortest path growth rate. In our numerical experiments we found that the asynchronous method outperformed the synchronous method on all moderately and weakly coupled problems and that this improvement was greatest for problems with more inter-processor connections. For the strongly coupled problems the rate of convergence of APJ was close to our shortest path performance guarantee and for the weakly coupled problems it was close to our weakest link performance barrier. Focusing

Table 9: Residual errors and bounds

	$\kappa(A)$ est'		(log 10)					
Matrix ID	sync'	async'	$\|\|b\|\|$	sync' res'	bound	async' res'	bound
a1	3.636	3.356	4.954	-3.793	-3.793	-4.015	-3.014
a2	20.00	18.60	3.956	-4.791	-4.791	-4.959	-4.012
a3	36.36	36.14	3.007	-5.742	-5.740	-5.841	-4.962
a4	39.60	39.39	2.616	-6.132	-6.130	-6.282	-5.352
a5	39.95	37.91	2.598	-6.164	-6.149	-6.762	-5.371
a6	39.84	39.36	2.597	-6.333	-6.150	-6.891	-5.372
a7	39.97	39.92	2.596	-6.361	-6.150	-6.917	-5.372
b1	3.636	3.251	4.914	-3.834	-3.833	-4.129	-3.055
b2	19.99	18.94	3.916	-4.832	-4.831	-5.043	-4.052
b3	36.36	36.15	2.974	-5.774	-5.773	-5.929	-4.994
b4	39.60	39.17	2.613	-6.134	-6.133	-6.381	-5.230
b5	39.95	37.21	2.597	-6.172	-6.149	-6.734	-5.371
b6	39.85	39.24	2.597	-6.338	-6.150	-6.901	-5.372
b7	39.97	39.92	2.597	-6.360	-6.150	-6.906	-5.247
c1	3.636	3.152	4.958	-3.791	-3.789	-4.124	-2.886
c2	20.00	18.73	3.960	-4.789	-4.787	-4.985	-3.884
c3	36.36	36.05	3.010	-5.737	-5.737	-5.904	-4.834
c4	39.60	39.45	2.616	-6.132	-6.130	-6.289	-5.227
c5	39.95	37.08	2.597	-6.161	-6.149	-6.690	-5.246
c6	39.84	39.51	2.597	-6.331	-6.150	-6.832	-5.247
c7	39.97	39.93	2.597	-6.361	-6.150	-6.814	-5.247
d1	3.636	3.095	4.955	-3.793	-3.791	-4.120	-2.888
d2	19.99	18.71	3.957	-4.791	-4.789	-5.025	-3.789
d3	36.36	36.15	3.008	-5.739	-5.739	-5.881	-4.960
d4	39.60	39.43	2.616	-6.132	-6.130	-6.292	-5.227
d5	39.95	39.21	2.598	-6.163	-6.149	-6.550	-5.246
d6	39.84	39.70	2.597	-6.333	-6.150	-6.668	-5.247
d7	39.97	39.93	2.596	-6.361	-6.150	-6.908	-5.247

on the weakly coupled problems where APJ performed well, our weakest link bound should inform the design of asynchronous linear solvers in two ways.

1. Load balancing. To maximize the rate of convergence of the slowest subsystem it is necessary to balance the workload between the different processors. If the diagonal block submatrices are all roughly the same, so that $\rho(M_{ii})$ is approximately constant for $i = 1, \ldots, N$, then this reduces to balancing the workload so that the different processors have, as closely as possible, the same update rate. If $\rho(M_{ii})$ varies widely then workload should be balanced to try to increase the update rate of the processors that iterate diagonal block submatrices with larger spectral radii, at the expense of the update rate of processors that iterate diagonal block submatrices with smaller spectral radii.

2. Communication can be skipped. In the domain of problems where the weakest link performance barrier determines the performance we can reduce the frequency with which processors broadcast their new states without harming the rate of convergence. By only broadcasting its state every ten iterations, say, the update rate of a processor can be increased, which will improve the rate of convergence. Allowing different rates of broadcast between different

processors could be one approach to load balancing. This would effectively increase the update rate whilst decreasing the shortest path rate.

Acknowledgements

The first author was supported by Engineering and Physical Sciences Research Council (EPSRC) grant EP/I005293 “Nonlinear Eigenvalue Problems: Theory and Numerics”. The second author was supported by EPSRC grant EP/I006702/1 “Novel Asynchronous Algorithms and Software for Large Sparse Systems”. This research made use of the Balena High Performance Computing (HPC) Service at the University of Bath.

We would like to thank our two referees, whose valuable comments and suggestions have considerably improved the original manuscript over several rounds of reviewing. We would also like to thank Dr Mark Muldoon (University of Manchester, UK) for many useful discussions.

References

1. H. Avron, A. Druinsky, and A. Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS ’14, pages 198–207, 2014.
2. D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
3. I. Bethune, J.M. Bull, N.J. Dingle, and N.J. Higham. Performance analysis of asynchronous Jacobi’s method implemented in MPI, SHMEM and OpenMP. *IJHPCA*, 28(1):97–111, 2014.
4. V.D. Blondel, M. Karow, V. Protassov, and F. R. Wirth. Special issue on the joint spectral radius: Theory, methods and applications. *Linear Algebra and its Applications*, 428(10), 2008. Special Issue on the Joint Spectral Radius: Theory, Methods and Applications.
5. J.M. Bull and T.L. Freeman. Numerical performance of an asynchronous Jacobi iteration. In *Parallel Processing*, volume 634 of *Lecture Notes in Computer Science*, pages 361–366. Springer Berlin Heidelberg, 1992.
6. D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and its Aplications*, 2:199–222, 1969.
7. N.J. Dingle and W.J. Knottenbelt. Distributed solution of large markov models using asynchronous iterations and graph partitioning. In *Proceedings of the 18th UK Performance Engineering Workshop*, pages 27–34, 2002.
8. 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*, 130:65 – 82, 1990.
9. A. Frommer and D.B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(12):201 – 216, 2000.
10. H. Furstenberg and H. Kesten. Products of random matrices. *Ann. Math. Statist.*, 31(2):457–469, 06 1960.
11. R. Gharavi and V. Anantharam. An upper bound for the largest lyapunov exponent of a markovian product of nonnegative matrices. *Theoretical Computer Science*, 332(1):543 – 557, 2005.
12. R. Horn and C. Johnson. Matrix analysis. *Cambridge University Press*, 1990.
13. S. Sridhar J. Liu, Stephen J. Wright. An asynchronous parallel randomized Kaczmarz algorithm. *arXiv:1401.4780*, 2014.

14. J. Lu and Y. Tang. Distributed asynchronous algorithms for solving positive definite linear equations over dynamic networks. *arXiv:1306.0260*, 2013.
15. F. Niu, B. Recht, C. R, and S.J. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.
16. J. N. Tsitsiklis and V. D. Blondel. The lyapunov exponent and joint spectral radius of pairs of matrices are hard—when not impossible—to compute and to approximate. *Mathematics of Control, Signals and Systems*, 10(1):31–40, 1997.
17. P. Walters. *An introduction to ergodic theory.* Graduate texts in mathematics. Springer, New York, Berlin, Heidelberg, 1982.