Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Journal of Computational and Applied Mathematics 176 (2005) 91–103
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

Asynchronous iterations with flexible communication: contracting operators

Didier El Baz[a], Andreas Frommer[b,*,1], Pierre Spiteri[c]

[a] *Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS, 7, avenue du Colonel Roche, F-31077 Toulouse CEDEX 4, France*
[b] *Fachbereich Mathematik und Naturwissenschaften, Universität Wuppertal, Gauss-Strasse 20 D-42097 Wuppertal, Germany*
[c] *Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique et d'Hydraulique de Toulouse, 2, rue Camichel, B.P. 7122, F-31071 Toulouse CEDEX 1, France*

Received 18 November 2003; received in revised form 2 June 2004

Abstract

The concept of flexible communication permits one to model efficient asynchronous iterations on parallel computers. This concept is particularly useful in two practical situations. Firstly, when communications are requested while a processor has completed the current update only partly, and secondly, in the context of inner/outer iterations, when processors are also allowed to make use of intermediate results obtained during the inner iteration in other processors.

In the general case of nonlinear or linear fixed point problems, we give a global convergence results for asynchronous iterations with flexible communication whereby the iteration operators satisfy certain contraction hypotheses. In this manner we extend to a contraction context previous results obtained for monotone operators with respect to a partial ordering.

MSC: 65Y05; 68Q10; 68Q22

Keywords: Asynchronous iterations; Parallel computing; Flexible communication; Fixed point methods

* Corresponding author.
E-mail addresses: elbaz@laas.fr (D. El Baz), frommer@math.uni-wuppertal.de (A. Frommer), pierre.spiteri@enseeiht.fr (P. Spiteri).

[1] The work of this author was partially funded by the French Ministry of Research through a visiting grant at ENSEEIHT.

0377-0427/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2004.07.009

1. Introduction

Parallel computers work efficiently only if the work load for a given computation between two synchronisation points can be distributed evenly among the processors. At a synchronisation point, processors generally need data which have been computed by other processors, so that usually they have to wait until the other processors have finished their computation and, in the case of distributed memory architectures, the communication of the data has been accomplished. There are situations where synchronisation may become a decisive bottleneck. For example, on supercomputers with several thousands of processors, synchronisation can become costly due to technical restrictions. Moreover in many applications, and in particular for nonlinear problems, it can be difficult to predict the computational cost of each parallel task, so that an even distribution of the work load between the processors cannot be achieved *a priori*. A similar situation arises if the parallel computer in use is a cluster of heterogeneous workstations which, in addition, may not be available in dedicated mode. Then, the computational power available on each processor is unpredictable, so that it is again impossible to obtain a fair assignation whereby each parallel task requires equal time between two synchronisation points. In such situations it can be advantageous to use the asynchronous paradigm instead of the synchronous one. We think in particular of iterative processes whereby each iterative step produces an approximation to the solution of a given problem. Then, classically, synchronisation will occur at the beginning of each iterative step where processors build up the value of the current iterate from the data computed in all processors. In the asynchronous case, these synchronisation points are completely skipped. Therefore, if certain processors perform their iterative step faster than others, then the processors will get 'out of phase'. When building up 'their' current iterate, the processors will now use data from other processors which will not correspond to the data used in the synchronized algorithm. In this manner, the asynchronous paradigm tends to eliminate idle times due to synchronisation. On the other hand, the resulting asynchronous iteration is less structured, and there is a need for theoretical results concerning the convergence and the speed of convergence of such methods.

Asynchronous iterations have been studied and implemented by many authors for a variety of different applications. Any attempt to list all relevant publications is beyond the scope of this paper. Instead, we refer to the overview article [11] and the book [5] for references in the case of linear and nonlinear systems of equations and minimization problems and the very recent papers [1–3] for multisplitting ideas and applications to complementarity problems. The recent paper [17] analyses for the first time asynchronous iterations from a stochastic perspective.

In this study, we further develop on a recent and general class of asynchronous iterations for linear and nonlinear fixed point problems which has first been brought forward in a mathematical form in [8,13]. This concept, called 'asynchronous iterations with flexible communication' allows for an even larger degree of freedom on when and how communications are to be performed than the classical model for asynchronous iterations (see [7,12,4,5]). In particular, the flexible communication model is well suited to the following two situations which we call the 'partial update' situation and the 'inner/outer' context.

In the partial update situation, each processor has several components, say a block, of the iterate vector to update, and it may happen that another processor requests data at a moment when this processor has updated only a part of the components of its block. In the classical asynchronous model, communication of data would have to be delayed until the update is completed. This actually introduces an undesirable partial synchronisation together with idle times. In the asynchronous model with flexible communication we avoid this drawback and allow for the possibility to communicate data as soon as it is requested, even if updates are completed only partly.