On asynchronous Newton-iterative methods

Matthias Pott*

Abstract

In [9] A.H. Sherman investigated Newton-iterative methods for nonlinear systems $F(x) = 0$. These are methods, where instead of the next Newton step, the linear system $F'(x(j))\, y = F'(x(j))\, x(j) - F(x(j))$ was not solved directly, but approximated by finite many steps of a linear iterative method. He showed, that the principle of many numerical methods, namely to start with rough and to end with fine approximations, which means here to increase the number of *inner* or *linear iterations*, leads to a significant speed up of convergence, assuming, that the evaluations of the function F and its partial derivatives are expensive. R.E. White [11] investigated the same problem by using a synchronized parallel multisplitting method for the inner iterations. We will introduce an asynchronous Newton-iterative method, which has the same properties in speed of convergence, but in addition, inner and outer iterations are totally decoupled, and moreover, the partial derivatives of F can be evaluated or approximated totally asynchronous. Interpreting the evaluation of component functions and partial derivatives as the most complex task, our method will converge superlinearly, and under suitable conditions at least quadratically.

Key words: Asynchronous iterations, Newton-iterative methods.

AMS(MOS) subject classification. 65J15.

1 Introduction

Baudet [1] extended asynchronous iterations, introduced by Chazan and Miranker [4] for linear systems, to nonlinear systems. Further he introduced *asynchronous iterations with memory,* where the operator of iteration depends on more than one of each components of already computed iterations. In the definition below, we will add the idea of a *pool* of iteration operators to this concept (compare [5]) and also that the real operators depend on parameters.

*Research supported by Sonderforschungsbereich 343: „Diskrete Strukturen in der Mathematik“, Universität Bielefeld. Postfach 100131, 33501 Bielefeld.

Definition 1.1 Let $\mathbb{K}$ be some set and

$$G^k : D_0^m \times D_h \subset \mathbb{R}^{nm} \times \mathbb{R}^p \to \mathbb{R}^n \,, \qquad \text{for all} \quad k \in \mathbb{K} \tag{1}$$

be a *pool* of operators, $\mathcal{K}$ be a sequence from $\mathbb{K}$, given by

$$k : \mathbb{N}_0 \to \mathbb{K} \,, \tag{2}$$

$\mathcal{S}$ be a sequence from $\mathbb{N}_0^{nm}$, denoted by

$$\begin{array}{rl} & s_i^l : \mathbb{N}_0 \to \mathbb{N}_0 \quad i = 1, ..., n \,,\; l = 1, ..., m \,, \\ \text{with} & s_i^l(j) \le j \quad \forall\, j \in \mathbb{N}_0 \,,\; l = 1, ..., m \,,\; i = 1, ..., n \end{array} \tag{3}$$

and $\mathcal{J}$ a sequence of subsets of $\{1, ..., n\}$:

$$J_j \subset \{1, ..., n\} \quad j = 0, 1, ... \,. \tag{4}$$

Finally, let $\mathcal{H}$ be a sequence from D_h,

$$h_j \in D_h \quad j = 0, 1, ... \tag{5}$$

and $x(0) \in D_0$ be an initial vector, then the iteration

$$x_i(j+1) := \begin{cases} x_i(j) & \text{, if } \; i \notin J_j \\ G_i^{k(j)}\left(\begin{pmatrix} x_1(s_1^1(j)) \\ \vdots \\ x_n(s_n^1(j)) \end{pmatrix}, ..., \begin{pmatrix} x_1(s_1^m(j)) \\ \vdots \\ x_n(s_n^m(j)) \end{pmatrix}, h_j \right) & \text{, if } \; i \in J_j \end{cases} \tag{6}$$

$$j = 0, 1, ... \,.$$

is called an *asynchronous iteration* .

Now we can denote an asynchronous iteration by $(\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{J}, \mathcal{H}, x(0))$. Of course, the problem to be solved, is determining a *common fixed point* ξ of the pool $\mathcal{G}$, e.g.

$$G^k(\xi, ..., \xi, h) = \xi \quad \forall\, k \in \mathbb{K} \,,\; h \in D_h \,,$$

where throughout this paper we will always assume that such a fixed point exists and is locally unique.

The concept of asynchronous parallelism we have in mind can be interpreted as a pure or a combination of two strategies: The idea of [4] was as follows: Each of the single processors of a multiprocessor, denoted by $\pi_1, ..., \pi_K$ retrieves some components of an iteration it belongs to from some shared memory, or the local memory of processors to which it is linked, computed at *time* steps $s_i^l(j)$, respectively.