Technical University of Chemnitz–Zwickau

Department of Mathematics

Joerg Weickert · Torsten Steidten

Efficient Time Step Parallelization of Full-Multigrid Techniques

Preprint-Reihe der Chemnitzer DFG-Forschergruppe
"Scientific Parallel Computing" SPC 94_1
January, 1994

Summary

This paper deals with parallelization methods for time–dependent problems where the time steps are shared out among the processors. A Full Multigrid technique serves as solution algorithm, hence information of the preceding time step and of the coarser grid is necessary to compute the solution at each new grid level. Applying the usual extrapolation formula to process this information, the parallelization will not be very efficient. We developed another extrapolation technique which causes a much higher parallelization effect. Test examples show that no essential loss of exactness appears, such that the method presented here shall be well–applicable.

1 Introduction

A lot of parallelization methods to achieve a faster computation of problems of mathematical physics have been introduced within the last few years. Most of them, the so–called Domain Decomposition (DD) methods, make use of a splitting of the calculation domain. To improve the computation of time-dependent calculations like parabolic initial boundary value problems another idea for parallelization was charging each processor with the computation of one certain time step. Applying multigrid techniques, the amount of work to be done at each processor will be considerable.
The proceeding is the following: Computation starts at the coarsest grid (1), that means, we employ a full–multigrid algorithm. Processor number one calculates the first time step now. When it has finished the first grid level, it can go on with grid number two, while processor two (for time step two) starts at the coarsest grid, making use of information of grid one, time step one:

$$\tilde{u}_1^2 := u_1^1.$$

Here u_q^j represents the solution and $\tilde{u}_q^j$ an approximate solution at grid level q, time step j. For grid number two, processor two needs results of grid two, time step one:

$$\tilde{u}_2^2 := u_2^1 + \cdots.$$

Besides, processor three is able to start with grid number one, getting results of processor two relative to this grid level:

$$\tilde{u}_1^3 := u_1^2,$$

and so on.
A generalized extrapolation formula for the approximate solution may be written in the form

$$\tilde{u}_{q+1}^{j+1} := u_{q+1}^j + I_q^{q+1}(u_q^{j+1} - u_q^j) \tag{1}$$

$j = 1, \cdots, k-1$ (time step)
$q = 1, \cdots, l-1$ (grid level)
I_q^{q+1} is an interpolation operator from grid level q to the next finer one.
Figure 1 illustrates this formula.

The whole process might be represented as a gradual one: Processor number one is computing the finest grid (l), processor number two is employed with the second finest ($l-1$), $\cdots$, processor number l is calculating the coarsest one (1), see figure 2.

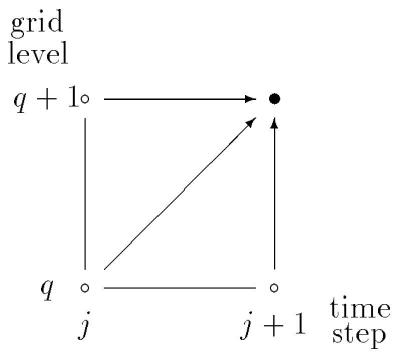

Figure 1: Usual extrapolation scheme

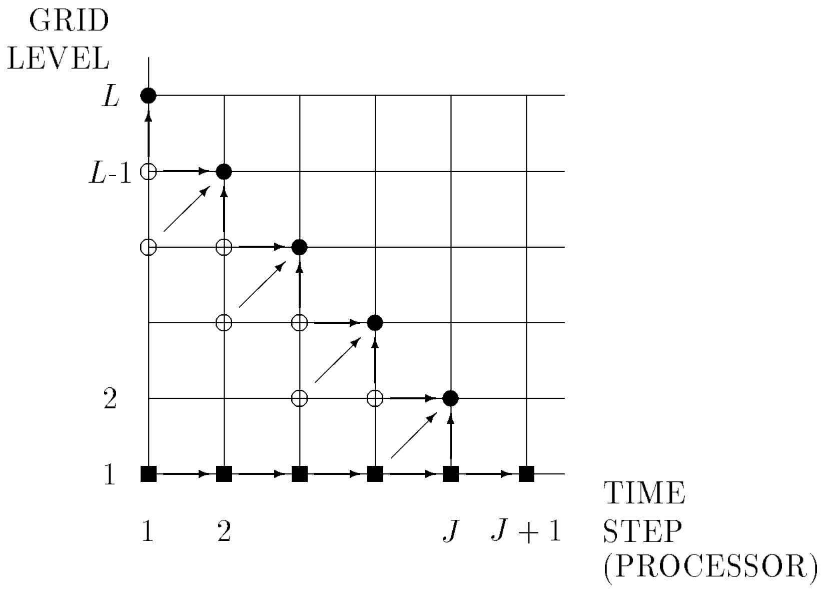

Figure 2: Multigrid process according to extrapolation (1)

That means, that the computations at the finest grid must be carried out successively. At this grid level no computation time can be saved by parallelization. This becomes clear by a comparision of computation times of the parallel and the sequential method:

- for parallel computing: $k \cdot t_l + t_{l-1} + \cdots + t_1$
- for successive computing: $k \cdot (t_l + t_{l-1} + \cdots + t_1)$

Profit of parallel computing: $(k-1) \cdot (t_{l-1} + \cdots + t_1)$

k	- number of time steps
l	- number of grid levels
t_q	- computation time at grid number q

But just at the finest grid the expense of computation time is much higher than at the coarser ones. For a two–dimensional model it is about four times as high as at the second finest grid. Consequently, it does not make sense to calculate at the coarser grids faster than at finer ones by choosing the number of processors greater than the number of grid levels to be considered, because the process has to wait for the solution at the finest grid. It is easy to be seen that no efficient parallelization is possible using this algorithm. Therefore a method towards a more efficient parallel computing of time–dependent full–multigrid techniques shall be developed.

2 A more suitable extrapolation formula

Formula (1) is obtained by a Taylor development which is presented here for the one–dimensional case (for more spatial dimensions it may be done analogously). The upper index represents the time step, the lower one the grid point. h is the step size of the current grid level, which we assume to be constant. Then the approximate value at a new grid level is calculated as follows:

$$\begin{aligned} u^{j+1}_{x_0+h} &= u^j_{x_0} + \frac{\partial u^j_{x_0}}{\partial x} \cdot h + \frac{\partial u^j_{x_0}}{\partial t} \cdot \tau + O(h^2+\tau^2) \\ &\approx u^j_{x_0} + \frac{u^j_{x_0+h} - u^j_{x_0}}{h} \cdot h + \frac{u^{j+1}_{x_0} - u^j_{x_0}}{\tau} \cdot \tau + O(h^2+\tau^2) \\ &= u^j_{x_0+h} + u^{j+1}_{x_0} - u^j_{x_0} + O(h^2+\tau^2) \end{aligned}$$

If $x_0 + h$ is a point of the current grid $q+1$, a point x_0 (distance h) will be a point of the next coarser grid q, such that we can consider values $u^*_{x_0}$ as solutions of this grid level, which leads to

$$\tilde{u}^{j+1}_{q+1} = u^j_{q+1} + I^{q+1}_q(u^{j+1}_q - u^j_q) \quad ,$$

i.e. formula (1) of chapter 1. (The lower index now denotes the grid level).
What prevents an efficient parallelization is the fact that the process has to wait for the solution of the preceding time step at the current grid, u^j_{q+1}. The corresponding term in the formula can be excluded by replacing the forward–difference–approximation of $\partial u^j_{x_0}/\partial x$ by a backward–difference–approximation. Since we reach a grid $(q-1)$ then which is two levels coarser than the current one $(q+1)$, we have to move a distance $2 \cdot h$ away from a point x_0 of grid q to meet a point of this grid $q-1$.
Then the backward–difference–approximation is

$$\frac{\partial u^j_{x_0}}{\partial x} \approx \frac{u^j_{x_0} - u^j_{x_0-2h}}{2h},$$

such that we have

$$\begin{aligned} u^{j+1}_{x_0+h} &\approx u^j_{x_0} + \frac{u^j_{x_0} - u^j_{x_0-2h}}{2h} \cdot h + \frac{u^{j+1}_{x_0} - u^j_{x_0}}{\tau} \cdot \tau + O(h^2+\tau^2) \\ &= \frac{1}{2}u^j_{x_0} + u^{j+1}_{x_0} - \frac{1}{2}u^j_{x_0-2h} + O(h^2+\tau^2) \end{aligned}$$

This corresponds to the new extrapolation formula

$$\tilde{u}^{j+1}_{q+1} := I^{q+1}_q(u^{j+1}_q + \frac{1}{2}u^j_q) - \frac{1}{2}I\!I^{q+1}_{q-1}u^j_{q-1}, \tag{2}$$

where $I\!I^{q+1}_{q-1}$ stands for an interpolation operator from grid $q-1$ to another one which is two levels finer.
The scheme of this extrapolation variant is shown by means of figure 3.

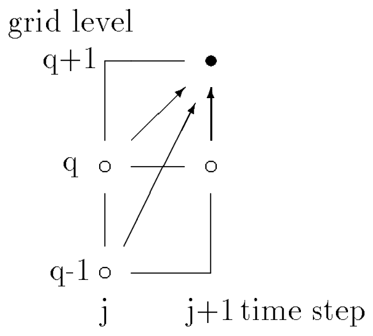

Figure 3: Extrapolation scheme without information of current grid level

3 The full parallel algorithm

Using the extrapolation formula of section 2, the proceeding does not make use of any information of former time steps concerning the current grid level, since u^j_{q+1} is replaced by information of coarser grids. In this way we get a "full parallel" algorithm, that means each processor is able to compute the same grid level. Information of other processors is needed from coarser grid levels only. The multigrid process for k time steps is shown by means of figure 4.

The processors work simultaneously from grid 3 onwards. At the two coarsest grids, the proposition of section 2 will not be applicable, because information of two coarser grid levels is necessary there. That's why the usual extrapolation (1) is employed here.
Towards a faster computation at the finest grid, the time step parallelization should be combined with methods like Domain Decomposition.
The advantage of a proceeding like this is that the multigrid algorithm of each time step is finished (almost) at the same time. However, the solution obtained in this way may be less exact than in the case of usual extrapolation. Therefore it is proposed to carry out one more multigrid step at the finest grid.

The entire process can be characterized as follows:

– At first computation with approximate values (less information, higher velocity);
– finally one more multigrid step at the finest grid for all time steps to receive the "exact" solution.

4 Numerical results

The numerical tests for the given results were carried out using the software package FEMGPM (Finite Element Multi–Grid Package) that originates in the program package implemented by M. Jung on an ESER 1040 computer under OS/ES in 1985 (cf. [1]) and further improved on an ESER 1056 under OS/VMS. This program was also installed on a VAX computer under UNIX/VMX in 1987. Since 1990 the FEMGPM software has been implemented on personal computers under MS–DOS or DR–DOS, respectively. FEMGPM

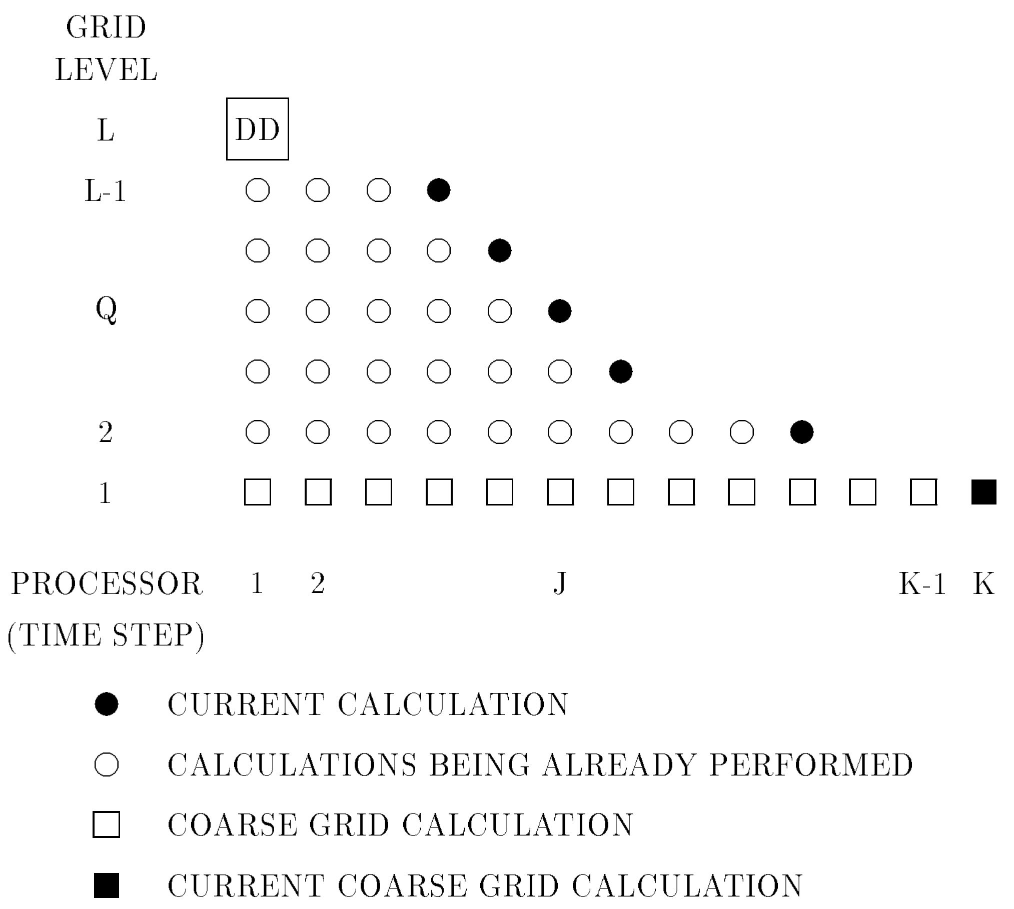
GRID LEVEL
L
DD
L-1
Q
2
1
PROCESSOR (TIME STEP)
1
2
J
K-1
K
● CURRENT CALCULATION
○ CALCULATIONS BEING ALREADY PERFORMED
□ COARSE GRID CALCULATION
■ CURRENT COARSE GRID CALCULATION

Figure 4: Proceeding of the full parallel algorithm

is implemented in FORTRAN. To overcome the existing limit of 640 kBytes under DOS systems the LAHEY FORTRAN compiler is used that allows to use the extended memory of the computers. That's why problems with a relatively high number of unknowns can be solved on personal computers. Nowadays also implementations on a transputer under PARIX and for SUN workstations under UNIX are available. FEMGPM is used as a test program for numerical algorithms as well as for solving practical problems. The user is able to deal with linear elliptic problems (e. g. heat equations or elasticity problems), with linear parabolic problems and with thermic–mechanically coupled problems. For more detailed information about FEMGPM the reader should study [2].

The new extrapolation method has been tested on some examples of the two–dimensional heat equation, and the results were compared with the ones of the usual extrapolation method (1). The calculation domain was a rectangle with a basic triangularization for the FEM-discretization as shown in figure 5. We considered the parabolic problem

$$\frac{\partial u}{\partial t} - \triangle u = \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(t,x,y)$$

with various right hand sides $f(t,x,y)$ which were chosen such that the exact solution is given analytically. So we were able to consider the difference between the exact (u) and the numerical solution $(\tilde{u})$ in the W_2^1-, L_2-, and $C-$ norms, respectively. The analytic solutions of our test examples were the following:

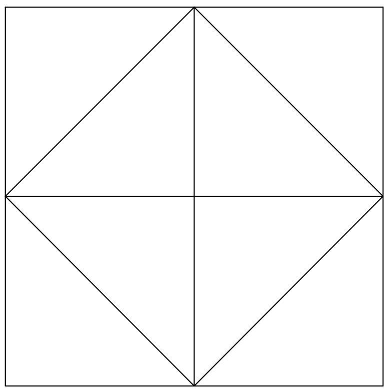

Figure 5: Calculation domain

Example 1: $u(t,x,y) = x(1-x)y(1-y)t^2$
Example 2: $u(t,x,y) = x(1-x)\sin\pi y \cos 2k\pi t$
Example 3: $u(t,x,y) = \sin\pi x \sin\pi y \cos 2k\pi t$

In examples 2 and 3 the parameter k controls how much the solution oscillates during the process. By this way we wanted to examine if higher oscillations have a stronger influence on the new (less exact) extrapolation formula.
The computations of the test examples were not carried out parallel but sequential, because we just wanted to consider the degree of exactness of our extrapolation method and not the effect on computation times. For all examples a Full Multigrid algorithm with five grid levels was employed, and 50 time steps were computed. The final multigrid step over all time steps (as proposed in section 3) was left out, it is not relevant for a comparison of the exactness of the two extrapolation variants.
As result we got the following error norms

$$\| u - \tilde{u} \|_*, \quad * \in \{W_2^1, L_2, C\} \; :$$

Example	Norm	Usual extrapolation (1)	New extrapolation (2)
1	W_2^1	$4.4213 \cdot 10^{-3}$	$4.4213 \cdot 10^{-3}$
	L_2	$3.7532 \cdot 10^{-5}$	$3.7803 \cdot 10^{-5}$
	C	$9.9596 \cdot 10^{-5}$	$1.0078 \cdot 10^{-4}$
2 (k=5)	W_2^1	$2.2316 \cdot 10^{-2}$	$2.2312 \cdot 10^{-2}$
	L_2	$3.2347 \cdot 10^{-3}$	$3.2336 \cdot 10^{-3}$
	C	$6.0517 \cdot 10^{-3}$	$6.0489 \cdot 10^{-3}$
2 (k=9)	W_2^1	$4.2243 \cdot 10^{-2}$	$4.2236 \cdot 10^{-2}$
	L_2	$8.5027 \cdot 10^{-3}$	$8.5016 \cdot 10^{-3}$
	C	$1.5784 \cdot 10^{-2}$	$1.5780 \cdot 10^{-2}$
2 (k=15)	W_2^1	$1.0388 \cdot 10^{-1}$	$1.0388 \cdot 10^{-1}$
	L_2	$2.2593 \cdot 10^{-2}$	$2.2593 \cdot 10^{-2}$
	C	$4.2555 \cdot 10^{-2}$	$4.2558 \cdot 10^{-2}$
3 (k=9)	W_2^1	0.1599	0.1599
	L_2	$3.2846 \cdot 10^{-2}$	$3.2842 \cdot 10^{-2}$
	C	$6.5403 \cdot 10^{-2}$	$6.5382 \cdot 10^{-2}$

The examples show that there is almost no difference between the error norms of both extrapolation techniques. Even higher oscillations are well assimilated by the new extrapolation method. The correspondence of the W_2^1–norms shows that also derivatives do not cause a deviation. So we can state that the technique developed in this paper may be used without any considerable loss of exactness. Applying parallel computing, a lot of computation time can therefore be saved at a reasonable price.

References

[1] M. Jung. Finite Element Multi–Grid Package FEMGP (November 1985 version). In G. Telschow, editor, *Second Multigrid Seminar, Garzau 1985*, pages 103–107, Berlin, 1986. Karl–Weierstrass–Institut. Report R–MATH–08/86.

[2] T. Steidten and M. Jung. Das Multigrid–Programmsystem FEMGPM zur Lösung elliptischer und parabolischer Differentialgleichungen einschließlich mechanisch–thermisch gekoppelter Probleme (Version 06.90). Programmdokumentation, Technische Universität Karl–Marx–Stadt, Sektion Mathematik, 1990.