The 19th Cambridge Workshop on Cool Stars, Stellar Systems, and the Sun
Edited by G. A. Feiden

A parallel numerical algorithm to solve linear systems of equations emerging from 3D radiative transfer

Viktoria Wichert,[1] Mario Arkenberg, [1] Peter H. Hauschildt [1]

[1] Hamburger Sternwarte, Universität Hamburg, Hamburg, Germany

Abstract

Highly resolved state-of-the-art 3D atmosphere simulations will remain computationally extremely expensive for years to come. In addition to the need for more computing power, rethinking coding practices is necessary. We take a dual approach by introducing especially adapted, parallel numerical methods and correspondingly parallelizing critical code passages. In the following, we present our respective work on PHOENIX/3D.
With new parallel numerical algorithms, there is a big opportunity for improvement when iteratively solving the system of equations $[E_n - \Lambda^*(1-\epsilon)]\, J_{new} = J_{fs} - \Lambda^*(1-\epsilon)J_{old}$ emerging from the operator splitting of the radiative transfer equation $J = \Lambda S$. The narrow-banded approximate Λ-operator Λ^*, which is used in PHOENIX/3D, occurs in each iteration step. By implementing a numerical algorithm which takes advantage of its characteristic traits, the parallel code's efficiency is further increased and a speed-up in computational time can be achieved.

1 Introduction

A vital part of computing atmosphere models, as well as a time consuming one, is finding a solution to the 3D radiative transfer equation numerically. To reduce the often extensive computation times, a parallel version of the PHOENIX/3D code was implemented in OpenCL and MPI (cf. Hauschildt & Baron (2011)). Figure 1 features the computational times needed for a 3D OpenCL radiative transfer calculation on several different devices, including a comparison between serial code, parallel MPI code and parallel OpenCL code. As can be seen from the plot, the parallel OpenCL version of PHOENIX/3D already achieves a significant speed-up of the calculations compared to the serial version, while being able to run on a wide range of devices, such as CPUs, GPUs and MICs. Since results of the parallelization are overall promising, it is reasonable to assume that further parallelizing the code will result in even smaller wall clock times. Instead of understanding parallelization as a pure programming problem, this paper will focus on suitable parallel algorithms.

The radiative transfer code consists of a formal solution step and an operator splitting step (see Kalkofen (1987)). The latter is one of the more time consuming parts of the 3D radiative transfer calculations, so any approach to further speed-up the code should primarily be focused on it.
The operator splitting step contains an iterative scheme to find the mean intensities J via the operator splitting equations $[1 - \Lambda^*(1-\epsilon)]\, J_{new} = J_{fs} - \Lambda^*(1-\epsilon)J_{old}$, where Λ^* is the approximate Λ-operator (ALO). There are several kinds of ALOs which all lead to a stable iterative scheme while keeping computational efforts small. PHOENIX/3D uses a narrow-banded $n \times n$ approximate Λ-operator (ALO) with a half bandwidth of $k << n$. Since this is an iterative scheme, similar systems with the same ALO Λ^* but different right-hand sides have to be solved.

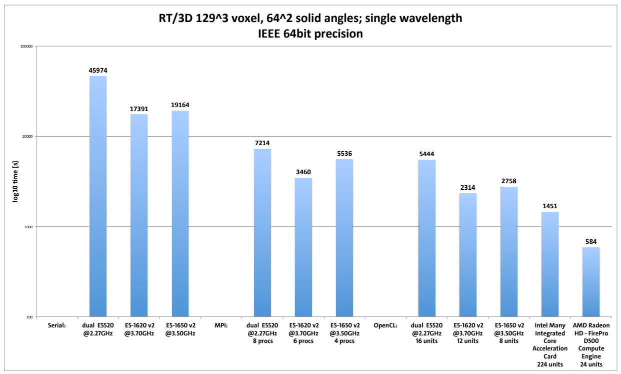


Figure 1: Computational time for a 3D OpenCL RT run with a single wavelength on a 129^3 voxels, 64^2 solid angles grid

In the current PHOENIX/3D version, the systems of equations are solved by a parallel sparse Jacobi solver. Replacing it by a parallel numerical solver which is especially suited to the features of Λ^* will result in a further decrease of computational time.

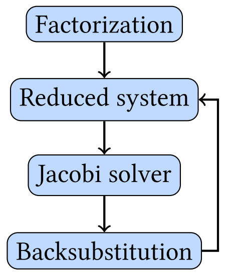
The following section, Methods and Implementation, will explain the algorithm's structure and details of how the before-mentioned characteristics of the problem are taken into account in the parallel solver. In Summary and Conclusion, results will be summarized and interpreted in the context of a successful implementation into the PHOENIX/3D code and its applicability to the radiative transfer problem.

2 Methods and Implementation

In this section the algorithm will be explained using the general systems of equations $Mx_i = b_i$, where M is a diagonally dominant, narrow-banded $n \times n$ matrix and $x_i, b_i \in \mathbf{R}^n \ \forall i$. The number of systems of equations with the same matrix M that have to be solved, i, is not fixed, but the algorithm will be the more effective the more systems are solved. Narrow-banded means that the half-bandwidth k fulfills the condition $k << n$. Diagonally dominant is defined by $\sum_{i \neq j} m_{ij} < m_{ii} \ \forall i \in \{1, \dots, n\}$. The latter assumption is made for the sake of simplicity. A system with a non diagonally dominant matrix can still be solved after minor changes to the algorithm (see next section).
The next section will explain how the algorithm solves the systems of equations effectively in parallel It is followed by remarks on the algorithm's implementation.

2.1 The Algorithm

The basic idea of this algorithm is to adapt the concept of the Gaussian elimination algorithm to parallel systems. The Gaussian algorithm can be described as follows: the matrix is factorized, which here means LU decomposed into $M = LR$. Afterwards there is a forward elimination step, which finds the solution z_i to the problem $L \underbrace{Rx_i}_{z_i} = b_i$. Then the solution to the original problem is computed by solving $Rx_i = z_i$ in a backwards substitution step.
The algorithm by Arbenz et al. (see Arbenz *et al.* (1999)) transfers this idea to solve systems of equations in parallel by spatially decomposing the matrix M. This specific decomposition only is interesting (in terms of computational effort) in cases where the matrix is narrow-banded. The second prerequisite, M being diagonally dominant, originates from the need to LU decompose parts of the original matrix. In case of a non diagonally dominant matrix M, it will be necessary to introduce additionally partial pivoting, since otherwise the LU decomposition might not be stable. The decomposition then takes the form $PM = LR$ instead of $M = LR$. Due to the additional pivoting matrix P, the overall algorithm has to be adapted accordingly in the non diagonally dominant case.
LU decomposing the whole matrix M, such as is done in the classical Gaussian algorithm, would be computationally expensive for big n. Therefore this algorithm finds a way where only one type of sub-matrices has to be LU decomposed. Since the 3D radiative transfer code solves similar systems (same matrix, different right hand sides) repeatedly, computing time can be decreased for sufficiently large enough problems by first factorizing the matrix, and only then solve the systems of equations, which is easier now with a factorized matrix.
In the first step of the algorithm, the factorization of M is done. Afterwards, the adapted RHS is computed to complete the so-called reduced system, which is then solved by a parallel Jacobi solver. The last step in computing the original system's solution is the back-substitution step. The general system behind the algorithm is also shown in the following flowchart.



Since the original narrow-banded matrix's entries are mostly zero, it is spatially decomposed into several types of sub-matrices A, B, C and D along the non-zero areas (see fig. 2). Indexes U and L indicate upper resp. lower diagonals. Each processing element then works on its part of the original matrix independently. Unlike the original back and forth-substitution algorithm, only sub-matrices of type A have to be LU decomposed.
Each processing element computes its part of the reduced system $S\xi_i = c_i$, where S is a $(p-1)k \times (p-1)k$ matrix (see fig. 3), whose structure is shown schematically by using $k \times k$ sub-matrices T, U and V. These new types of sub-matrices are calculated from L, R, B, C and D. Overlapping areas in the figure indicate data exchange between processing elements. Note that this is the first time since distributing work to the processors that communication has to take place between processing elements. The reduced matrix S can then be used to solve all the similar systems that were mentioned before. Only the RHS b_i has to be manipulated accordingly for every new system and is then denoted as c_i. The reduced system $S\xi_i = c_i$ is then solved in parallel by an implementation of the classic Jacobi solver. Afterwards, ξ_i is used to compute the original system's solution x_i via back-substitution. Every linear system containing the same matrix M can now be solved accordingly by computing the reduced system's solution and doing a back-substitution.

In relation to the linear systems emerging from the radiative transfer equation this means that during the first iteration step, the original matrix $M = E_n - \Lambda^*(1-\epsilon)$ is factorized. During all other iteration steps, only the reduced linear system has to be solved after adapting the current RHS.

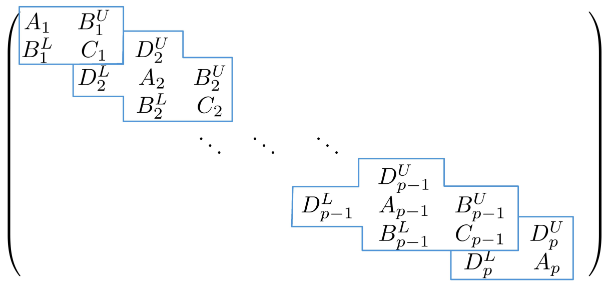


Figure 2: Decomposition of the original $n \times n$ matrix M onto p processing elements

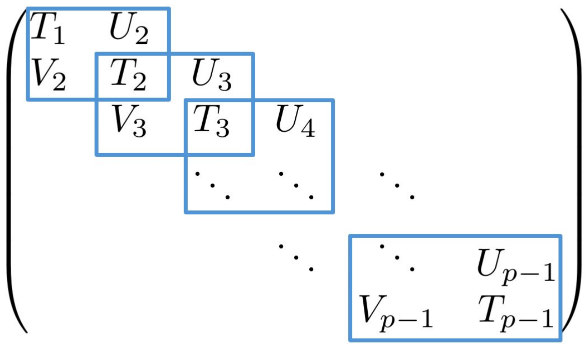


Figure 3: The resulting reduced system S consisting of $k \times k$ blocks

2.2 Remarks on Implementation

There is more than one option to use PHOENIX/3D in parallel, but there are several reasons to start implementing a new algorithm for PHOENIX/3D in OpenCL. First of all, OpenCL code runs on a range of devices, such as CPUs, GPUs, MICs and combinations thereof. It is an open platform, so the code can be used independently of the choice of a certain manufacturer's devices. Figure 1 also suggests that the OpenCL radiative transfer code might have a slight advantage in timing over the MPI version. Furthermore, there is the option to adapt the code to run as a OpenCL/MPI hybrid later on.

The code is started on the CPU (host) and distributes parallel tasks to the chosen device via kernels. In this algorithm, the parallel tasks are subdivided into four kernels (see flowchart): the first one does the factorization of the original matrix and only is executed once per wavelength. It includes spatially distributing the matrix elements to the processing elements by assigning each processing element its own id and distributing matrix elements according to their indexes. From here, the second kernel adapts the current RHS and therefore completes the reduced system, which is now solved by the third kernel, a parallel Jacobi solver. Afterwards, the back-substitution kernel computes the original system's solution from the reduced system's in the fourth kernel. The last three steps are now repeated for every similar system of equations.

Apart from few exceptions, there is no need for interprocessor communication. Only during the last step of the factorization while computing the reduced matrix S, data has to be shared between neighboring processing elements (see fig. 3).

The parallel algorithm has a higher memory demand than the serial one, since the different sub-matrices are allocated in addition to the original matrix. Although, after computing them, M is not needed for the remainder of the algorithm.

In case p equals one, no reduced matrix S exists. Instead, there is a shortcut implemented to solve the systems of equations directly via a serial Jacobi solver.

3 Summary and Conclusions

Extensive testing of PHOENIX/3D's parallel 3D radiative transfer code has shown that it already achieves a considerable speed-up compared to the serial version. To further decrease computational effort, a parallel algorithm has been introduced to solve the linear systems of equations emerging from the 3D radiative transfer algorithm.

Since the algorithm is not yet fully tested in 3DRT, there are no test results yet. Still, there are reasons to assume that usage of Arbenz' algorithm together with PHOENIX/3D is worthwhile. Especially in cases where the number of iterations in the operator splitting step is high (e.g. strong scattering), the computational costs of factorizing the original matrix can be compensated by saving computation time when solving the smaller, reduced system numerous times. Nevertheless, the overhead created by decomposing the matrix and transferring data to the different processing elements is only worth the effort if the original system is large enough, i.e. if the overhead is covered up by saving time doing the actual solving in parallel.

In cases where these preconditions are met, the algorithm will further decrease the 3D radiative transfer code's wall clock time.

References

Arbenz, P., Cleary, A., & Dongarra, J. 1999, EuroPar '99 Parallel Processing.

Hauschildt, P. & Baron, E. 2011, A&A, 533.

Kalkofen, W. 1987, *Numerical Radiative Transfer*.