Efficient Parallelization of the Schroedinger Equation

Nicolas Mattia

Winter Semester 2013-2014

Semester Project

at the
Department of Information Technology
and Electrical Engineering
ETHZ

Professor: Prof. Dr. Mathieu Luisier Supervisor: Sascha Brück

January 29, 2014
Zurich

Contents

Abstract

The Retarded Green's Function is a method used for solving quantum mechanical problems, for instance those arising when quantum mechanical effects have to be taken into account in nanoscale devices. This report presents an algorithm that was developed in order to efficiently parallelize a way of computing the Retarded Green's Function. The Retarded Green's Function and the Block Cyclic Reduction are presented as a theoretical background for the aforementioned algorithm. The algorithm itself is then presented. The last part focuses on tests that measured the computational time of the algorithm ran as a Matlab script, as well as improvement ideas.

Chapter 1

Introduction and Motivation

Almost fifty years ago, Gordon E. Moore noticed what would be later known as the famous Moore's law : the number of transistors on integrated circuits doubles approximately every two years. That fact led to the improvement of the functionality of a large number of electronics systems, like audio players, televisions, personal computers, and cellular phones. Now the point has been reached where that downscale comes with a new problem : it is not possible to neglect the quantum mechanical effects in the ever smaller nanoscale devices' characteristics anymore.

For that reason, it has now become important to trade the previously used methods with semi-classical approaches like the drift-diffusion model and Monte Carlo for more accurate ones like the Non-equilibrium Green's Function (NEGF). Those new methods demand more computational power, but the recent progress of computational capabilities, like the rise of the number of supercomputers and their efficiency, brings new possibilities. It is becoming cheaper and quicker to simulate nanodevices rather than building them in order to get an idea of their characteristics. This project focused a new way of parallelizing the resolution of the Retarded Green's Function.

Green's Function The method that was used in this project, the Retarded Green's Function, is part of the Non-equilibrium Green's Function (NEGF) formalism, more precisely its formalism for discrete setups which takes a matrix form. The Retarded Green's Function will take the form of a matrix whose number of blocks grows with the size of the system we are trying to simulate. Since new nanodevices require an atom level description, the aforementioned matrix can contain thousands or millions of elements.

OMEN The OMEN suite is an electronic simulation tool that uses the sp3d5s* semi-empirical tight-binding method. Thanks to multiple parallelism levels, it can benefit from and fully stress the largest supercomputers available. The present project's goal was to develop and benchmark a new

method of parallelisation for OMEN's way of solving of Retarded Green's Function problems.

Outline This report presents the algorithm that was developed in order to efficiently parallelize the way of computing the Retarded Green's Function. The following chapter will present the theoretical background used in the algorithm. Then, the algorithm itself will be presented, where storage and complexity questions will be addressed as well. Chapter 4 presents and discusses the results of time measuring tests that have been conducted on a Matlab implementation of the algorithm.

Chapter 2

Background

2.1 The Block Cyclic Reduction (BCR)

The so called Block Cyclic Reduction (*BCR*) is a method for reducing a block tridiagonal system to an equivalent one of smaller size. Let us define a tridiagonal system as following :

$$M = \begin{pmatrix} d_1 & u_1 & 0 & \cdots & 0 \\ l_1 & d_2 & u_2 & \ddots & \vdots \\ 0 & l_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & d_{N-1} & u_{N-1} \\ 0 & \cdots & 0 & l_{N-1} & d_N \end{pmatrix} \tag{2.1}$$

The reduction creates a new system $\tilde{M}$ with the blocks $\tilde{M}_{k,k}$, $\tilde{M}_{k,k-1}$, $\tilde{M}_{k,k+1}$, $\tilde{M}_{k-1,k}$ and $\tilde{M}_{k+1,k}$ equal to 0, $\tilde{M}_{k-1,k-1}$ and $\tilde{M}_{k+1,k+1}$ modified, and two new non-zero ones: $\tilde{M}_{k+1,k-1}$ and $\tilde{M}_{k-1,k+1}$. The two latter ones represent a new connection between the separated ones $\tilde{M}_{k-1,k-1}$ and $\tilde{M}_{k+1,k+1}$. Two temporary matrices X and Y are used

$$X = d_k^{-1} l_{k-1} = M_{k,k}^{-1} M_{k,k-1} \tag{2.2}$$

$$Y = d_k^{-1} u_k = M_{k,k}^{-1} M_{k,k+1}; \tag{2.3}$$

with which the new blocks are formed

$$\tilde{d}_{k-1} = d_{k-1} - u_{k-1} X \tag{2.4}$$

$$\tilde{d}_{k+1} = d_{k+1} - l_k Y \tag{2.5}$$

$$\tilde{u} = -u_{k-1} Y \tag{2.6}$$

$$\tilde{l} = -l_k X \tag{2.7}$$

$$\begin{pmatrix} \ddots & \ddots & & & \\ \ddots & d_{k-1} & u_{k-1} & & \\ & l_{k-1} & d_k & u_k & \\ & & l_k & d_{k+1} & \ddots \\ & & & \ddots & \ddots \end{pmatrix} \equiv \begin{pmatrix} \ddots & \ddots & & & \\ \ddots & \tilde{d}_{k-1} & 0 & \tilde{u} & \\ & 0 & 0 & 0 & \\ & \tilde{l} & 0 & \tilde{d}_{k+1} & \ddots \\ & & & \ddots & \ddots \end{pmatrix} \tag{2.8}$$

If this is done for $k = \{2, 4, 6, ..., N-1\}$, and then the zeros between our newly created elements are removed, the system takes the form of a tridiagonal matrix in which we can eliminate the even elements anew.

$$\begin{pmatrix} \ddots & \ddots & & & \\ \ddots & \tilde{d}_{k-1} & 0 & \tilde{u} & \\ & 0 & 0 & 0 & \\ & \tilde{l} & 0 & \tilde{d}_{k+1} & \ddots \\ & & & \ddots & \ddots \end{pmatrix} \rightarrow \begin{pmatrix} \ddots & \ddots & & \\ \ddots & \tilde{d}_{k-1} & \tilde{u} & \\ & \tilde{l} & \tilde{d}_{k+1} & \ddots \\ & & \ddots & \ddots \end{pmatrix} \tag{2.9}$$

It is then possible to repeat this procedure (removing the even elements, and then removing the zero-entries) to end up with a 2×2 completely block-cyclic-reduced block matrix.

2.2 The Retarded Green's Function (RGF)

Quantum mechanical problems are described by Schroedinger's equation

$$i\hbar \frac{\partial}{\partial t}\psi = \hat{H}\psi \tag{2.10}$$

Since it is difficult to find solutions for the time dependent case, it is common to find solutions for the time independent case (or steady-state). One method for solving the steady-state case is to use the Green's Function, which is represented by a matrix in the discrete case. The diagonal elements of that matrix are then directly proportional to the wave function of the system we are trying to simulate. That method will shortly be described here, but only as applied to our case.

2.2.1 General RGF

The Green's Function, G^R, is in our case defined as follows :

$$(E - H - \Sigma^R_{bound} - \Sigma^R_{scatt})G^R = I \tag{2.11}$$

where

- E is the diagonal matrix of the energy level we are trying to find a solution for
- H is the Hamiltonian of the system
- Σ^R_{bound} is the boundary self-energy matrix
- Σ^R_{scatt} is the scattering self-energy matrix, a block tridiagonal matrix

To simplify the notation, we can write

$$D = E - H - \Sigma^R_{scatt} \tag{2.12}$$

In a tight-binding model, to which the rest of this report is limited, the matrix H takes a block tridiagonal form. Now, since in this case both Σ^R_{scatt} and H are block tridiagonal, and E is diagonal, the matrix D is block tridiagonal as well. Lastly, the matrix Σ^R_{bound} has only two non zero blocks, located at the top-left and bottom-right corners

$$\Sigma^R_{bound} = \begin{pmatrix} \Sigma^{R(L)}_{bound} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \Sigma^{R(R)}_{bound} \end{pmatrix} \tag{2.13}$$

which means that the problem is reduced to finding $G^R = (D - \Sigma^R_{bound})^{-1}$, the inverse of a block tridiagonal matrix.

The first step for computing G^R is to compute the Green's Function first step g^R, which it is possible to compute in a recursive way. The block diagonal elements of G^R are sufficient for recovering the wave function. From here on, and for an easier notation, the Green's function G^R's and the Green's function's first step g^R's diagonals will simply be refered to as G and g, respectively.

For the last block element of g, the following relation holds

$$g_N = (d_N - \Sigma^{R(R)}_{bound})^{-1} \tag{2.14}$$

where N is the number the block diagonal elements of the matrix D. For two contiguous elements in g the following backward recursion is used:

$$g_k = d_k - u_k g_{k+1} l_k, k = \{N-1, N-2, ..., 1\} \tag{2.15}$$

and similarily for initiating G

$$G_1 = (d_1 - u_1 g_2 l_1 - \Sigma^{R(L)}_{bound})^{-1} \tag{2.16}$$

and then a forward recursion is used:

$$G_k = g_k + g_k u_{k-1} G_{k-1} l_{k-1} g_k, k = \{2, 3, ..., N\} \tag{2.17}$$

d_k, u_k and l_k the elements of D as defined in 2.1 for a tridiagonal system.

2.2.2 RGF with partially reduced systems

The first element of g and the first and last elements of G for a system and its reduced equivalent will be identical. It means we can write

$$g = \begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix}, \hat{g} = \begin{pmatrix} \hat{g}_1 \\ \hat{g}_2 \end{pmatrix} \tag{2.18}$$

and

$$G = \begin{pmatrix} G_1 \\ \vdots \\ G_N \end{pmatrix}, \hat{G} = \begin{pmatrix} \hat{G}_1 \\ \hat{G}_2 \end{pmatrix} \tag{2.19}$$

with

$$g_1 = \hat{g}_1 \tag{2.20}$$

$$G_1 = \hat{G}_1, G_N = \hat{G}_2 \tag{2.21}$$

or, for the g and G corresponding to a system D that has been split in N_{CN} (N_{CN} is the number of computing nodes, see next chapter) subsystems $D_{\{n\}}$ (with $n = \{1, 2, ..., N_{CN}\}$). Each of these N_{CN} subsystems $D_{\{n\}}$ will now be completely reduced to N_{CN} subsystems $\tilde{D}_{\{n\}}$ with corresponding $\tilde{g}$ and $\tilde{G}$. Some of the elements of g and G will then be equal to some of the elements of $\tilde{g}$ and $\tilde{G}$, according to (2.20) and (2.21)

$$g = \begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix} = \begin{pmatrix} \tilde{g}_{\{1\},1} \\ g_2 \\ \vdots \\ g_{\frac{N}{N_{CN}}} \\ \tilde{g}_{\{2\},1} \\ g_{\frac{N}{N_{CN}}+2} \\ \vdots \\ g_{\frac{nN}{N_{CN}}} \\ \tilde{g}_{\{n+1\},1} \\ \vdots \\ g_N \end{pmatrix} \tag{2.22}$$

and

$$G = \begin{pmatrix} \tilde{G}_{\{1\},1} \\ G_2 \\ \vdots \\ G_{\frac{N}{N_{CN}}-1} \\ \tilde{G}_{\{1\},2} \\ \tilde{G}_{\{2\},1} \\ G_{\frac{N}{N_{CN}}+2} \\ \vdots \\ \tilde{G}_{\{n-1\},2} \\ \tilde{G}_{\{n\},1} \\ \vdots \\ \tilde{G}_{\{N_{CN}\},2} \end{pmatrix} \tag{2.23}$$

Computing g and G with gaps Now considering those reduced subsystems $\tilde{D}_{\{n\}}$ of size 2×2, and assuming that $g_{\{n+1\},1}$ is known (for instance recovered from $\tilde{g}$, as seen in the previous paragraph), it is possible to compute $g_{\{n\},1}$ using the following formula :

$$M = \left(\tilde{D}_{\{n\}} - \begin{pmatrix} 0 & 0 \\ 0 & u_{\frac{nN}{N_{CN}}} g_{\{n+1\},1} l_{\frac{nN}{N_{CN}}} \end{pmatrix} \right)^{-1} \tag{2.24}$$

$$g_{\{n\},1} = M_{1,1} \tag{2.25}$$

where $u_{\frac{nN}{N_{CN}}}$ and $l_{\frac{nN}{N_{CN}}}$ are the upper and lower diagonal blocks between the blocks $\tilde{D}_{\{n\}}$ and $\tilde{D}_{\{n+1\}}$. Or, if there is no $g_{\{n+1\},1}$ because we are looking for results in the last subsystem, with $n = N_{CN}$,

$$M^{\Sigma(R)} = \left(\tilde{D}_{\{N_{CN}\}} - \begin{pmatrix} 0 & 0 \\ 0 & \Sigma^{R(R)}_{bound} \end{pmatrix} \right)^{-1} \tag{2.26}$$

$$g_{\{N_{CN}\},1} = M^{\Sigma(R)}_{1,1} \tag{2.27}$$

Similarily for G, it is possible to compute $G_{\{n\},2}$ with the knowledge of $G_{\{n-1\},2}$ and $g_{\{n+1\},1}$:

$$M = \left(\tilde{D}_{\{n\}} - \begin{pmatrix} 0 & 0 \\ 0 & u_{\frac{nN}{N_{CN}}} g_{\{n+1\},1} l_{\frac{nN}{N_{CN}}} \end{pmatrix} \right)^{-1} \tag{2.28}$$

$$G_{\{n\},2} = M_{2,2} + M_{2,1} u_{\frac{(n-1)N}{N_{CN}}} G_{\{n-1\},2} l_{\frac{(n-1)N}{N_{CN}}} M_{1,2} \tag{2.29}$$

where $u_{\frac{(n-1)N}{N_{CN}}}$ and $l_{\frac{(n-1)N}{N_{CN}}}$ are the upper and lower diagonal blocks between the blocks $\tilde{D}_{\{n-1\}}$ and $\tilde{D}_{\{n\}}$, and if there is no $n-1$ because $n=1$

$$M^{\Sigma(L)} = \left(\tilde{D}_{\{n\}} - \begin{pmatrix} \Sigma_{bound}^{R(L)} & 0 \\ 0 & u_{\frac{N}{N_{CN}}} g_{\{2\},1} l_{\frac{N}{N_{CN}}} \end{pmatrix} \right)^{-1} \tag{2.30}$$

$$G_{\{1\},2} = M_{2,2}^{\Sigma(L)}; \tag{2.31}$$

Chapter 3

Algorithm

3.1 Presentation of the algorithm

3.1.1 First Step: Reduction

The system's matrix $D = E - H - \Sigma^R_{scatt}$ presented in the previous chapter has a block triagonal form, meaning that a block cyclic reduction can be applied (*BCR*, 2.1). For parallelism purposes, it is interesting to split the system's matrix into N_{CN} submatrices, N_{CN} being the number of computing nodes.

$$D = \begin{pmatrix} d_1 & u_1 & 0 & \cdots & 0 \\ l_2 & d_2 & u_2 & \ddots & \vdots \\ 0 & l_3 & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & u_{N-1} \\ 0 & \cdots & 0 & l_{N-1} & d_N \end{pmatrix} \tag{3.1}$$

$$= \begin{pmatrix} D_{\{1\}} & U_{\frac{N}{N_{CN}}} & & & & & \\ L_{\frac{N}{N_{CN}}} & D_{\{2\}} & U_{\frac{2N}{N_{CN}}} & & & & \\ & L_{\frac{2N}{N_{CN}}} & \ddots & \ddots & & & \\ & & \ddots & D_{\{n\}} & U_{\frac{nN}{N_{CN}}} & & \\ & & & L_{\frac{nN}{N_{CN}}} & \ddots & \ddots & \\ & & & & \ddots & D_{\{N_{CN}\}} \end{pmatrix} \tag{3.2}$$

$$\tag{3.3}$$

where

$$L_{\frac{nN}{N_{CN}}} = \begin{pmatrix} 0 & \cdots & 0 & l_{\frac{nN}{N_{CN}}} \\ 0 & \cdots & 0 & 0 \\ \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{pmatrix}, U_{\frac{nN}{N_{CN}}} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \\ u_{\frac{nN}{N_{CN}}} & 0 & \cdots & 0 \end{pmatrix} \tag{3.4}$$

The submatrices

$$D_{\{n\}} = \begin{pmatrix} d_{a(n)} & u_{a(n)} & 0 & \cdots & 0 \\ l_{a(n)} & d_{a(n)+1} & u_{a(n)+1} & \ddots & \vdots \\ 0 & l_{a(n)+1} & d_{a(n)+2} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & u_{a(n+1)-2} \\ 0 & \cdots & 0 & l_{a(n+1)-2} & d_{a(n+1)-1} \end{pmatrix}, \tag{3.5}$$

$$a(n) = (n-1)\frac{N}{N_{CN}} + 1, n = \{1, 2, ..., N_{CN}\} \tag{3.6}$$

fulfill that purpose, and are tridiagonal too, which means that the *BCR* is indeed embarassingly parallel. The only thing to do is to send each subsystem $D_{\{n\}}$ to a different computing node n for a local *BCR*, and rearrange the results in a partially reduced matrix $\hat{D}$.

$$\hat{D} = \begin{pmatrix} \ddots & \ddots & & & & & & & \\ \ddots & \tilde{d}_{\{n-1\},2} & u_{\frac{(n-1)N}{p}} & & & & & & \\ & l_{\frac{(n-1)N}{p}} & \tilde{d}_{\{n\},1} & 0 & \cdots & 0 & \tilde{u}_n & & \\ & & 0 & 0 & \cdots & 0 & 0 & & \\ & & \vdots & \vdots & \ddots & \vdots & \vdots & & \\ & & 0 & 0 & \cdots & 0 & 0 & & \\ & & \tilde{l}_n & 0 & \cdots & 0 & \tilde{d}_{\{n\},2} & u_{\frac{nN}{p}} & \\ & & & & & & l_{\frac{nN}{p}} & \tilde{d}_{\{n+1\},1} & \ddots \\ & & & & & & & \ddots & \ddots \end{pmatrix} \tag{3.7}$$

The resulting matrix - or submatrices - are now rewritten in such a way that the new system is, again, tridiagonal. The non-zero elements are brought together and the zero elements that were separating them not taken

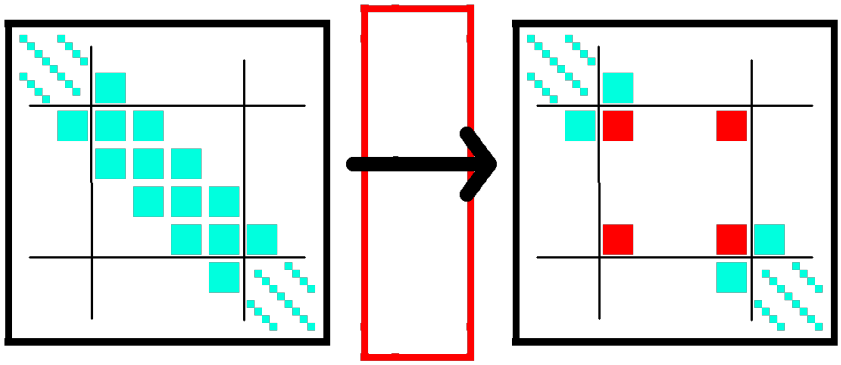

Figure 3.1: A graphic representation of a partially reduced matrix

into account.

$$D^{(2)} = \begin{pmatrix} \ddots & \ddots & & & & \\ \ddots & \tilde{d}_{\{n-1\},2} & u_{\frac{(n-1)N}{N_{CN}}} & & & \\ & l_{\frac{(n-1)N}{N_{CN}}} & \tilde{d}_{\{n\},1} & \tilde{u}_n & & \\ & & \tilde{l}_n & \tilde{d}_{\{n\},2} & u_{\frac{nN}{N_{CN}}} & \\ & & & l_{\frac{nN}{N_{CN}}} & \tilde{d}_{\{n+1\},1} & \ddots \\ & & & & \ddots & \ddots \end{pmatrix} \tag{3.8}$$

The new matrix $D^{(2)}$ is of size $\tilde{N} \times \tilde{N}$, with $\tilde{N} = 2N_{CN}$. The reduction step that was just presented can actually be repeated as long as the reduced matrix contains enough blocks for a reduction, that is as long as the resulting system is not smaller than 2×2 (as discussed later, in our case the system will only be reduced down to two 2×2 block matrices).

Stages and reduction steps This reduction has been implemented in stages. A stage i loads the matrix that was created at the reduction step $i-1$, and distributes it to $N_{CN}^{(i)}$ different nodes for a new reduction step i. $D^{(i)}$ is then the system in its i^{th} reduction stage, that is the system after the reduction step $i-1$. It will be of size $2N_{CN}^{(i-1)} \times 2N_{CN}^{(i-1)}$, $N_{CN}^{(i-1)}$ the number of computing node used in the reduction step $i-1$. In this algorithm, $N_{CN}^{(i)} = \frac{N_{CN}^{(i-1)}}{2}$, which assures that the system at stage i is big enough for being distributed on the $N_{CN}^{(i)}$ nodes. After the reduction, each node will store the result locally, for a later use in the reconstruction (see next section 3.1.2).

There will be in total N_S stages and N_R reduction steps, which both depend on N_{CN}. The reason to that is that only half the nodes that were

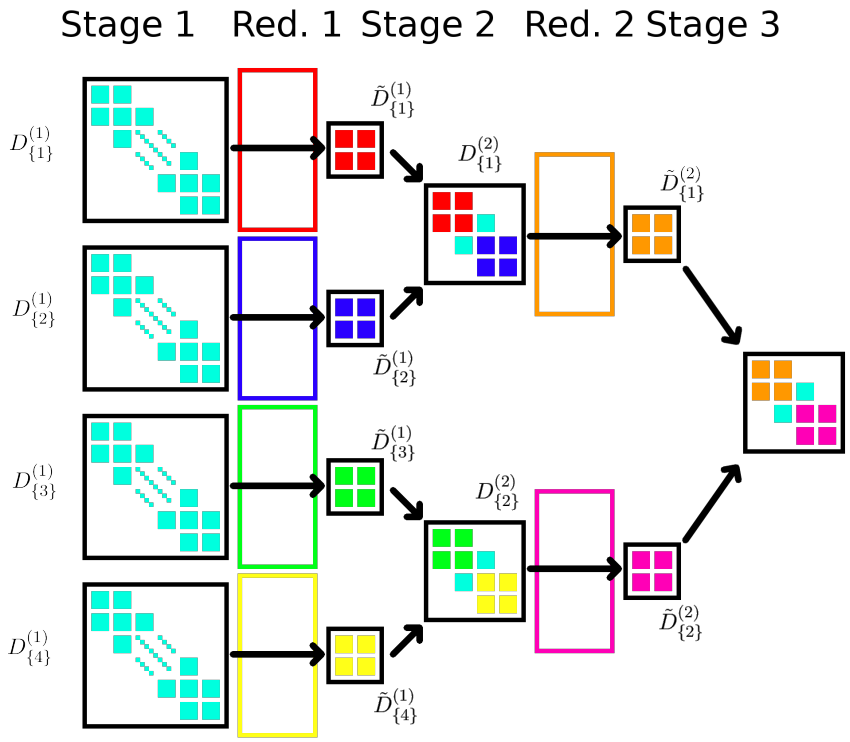

Figure 3.2: As an example, a reduction with $N_{CN} = 4$

used in the reduction step $i-1$ are used in the reduction step i. For $i = N_R$ there are only two nodes used, yielding our final 4×4 block matrix. The formulas for N_S and N_R are given later in this chapter.

3.1.2 Second Step: Reconstruction

The general idea in the reconstruction is to use the g elements of the last stage N_S to reconstruct those of stage $N_S - 1$, and then using the ones of stage $N_S - 1$ for $N_S - 2$, and so on. At first it will be shown that using only

the odd elements of $g^{(i+1)}$ it is possible to reconstruct the odd elements of $g^{(i)}$. It will be then shown that these odd elements of g of at stage i and the complete $G^{(i+1)}$ are sufficient to compute all the elements of $G^{(i)}$.

As a visual representation

- $\square$ is a stage
- $\dashrightarrow$ is a reduction step
- $\dashleftarrow$ is a reconstruction step

This is the reduction phase, starting from $\square_1$

$$\square_1 \quad \dashrightarrow_1 \quad \square_2 \quad \dashrightarrow_2 \quad ... \quad \dashrightarrow_{N_R-1} \quad \square_{N_S-1} \quad \dashrightarrow_{N_R} \quad \square_{N_S} \tag{3.9}$$

and this the reconstruction phase, starting from $\square_{N_S}$

$$\square_1 \quad \dashleftarrow_1 \quad \square_2 \quad \dashleftarrow_2 \quad ... \quad \dashleftarrow_{N_R-1} \quad \square_{N_S-1} \quad \dashleftarrow_{N_R} \quad \square_{N_S} \tag{3.10}$$

At each stage i we are going to use the reduced system $D^{(i)}$ stored on $N_{CN}^{(i)}$ nodes and the formulas of the section 2.2 on the Green's Function on partially reduced system to reconstruct $g^{(i)}$ and $G^{(i)}$. The main difference with the reduction is that the reconstruction does not start from the original system, but from the reduced one. The start point is then the subsystem $D^{(N_S)}$ stored on two nodes.

As mentioned before, $N_{CN}^{(i-1)} = \frac{N_{CN}^{(i)}}{2}$. The nodes at stage i can be regrouped in $\frac{N_{CN}^{(i)}}{2}$ pairs. The nodes within a pair will work sequentially, but a pair can work independently from the other pairs. A pair of nodes $\{n, n+1\}$, at stage i will, at the end of a reconstruction stage, yield two four-element arrays $g_{\{n,n+1\}}^{(i)}$ and $G_{\{n,n+1\}}^{(i)}$. In this case, the elements that will be reconstructed for $g^{(i)}$ are the odd ones

$$g_{\{n,n+1\}}^{(i)} = \begin{pmatrix} g_{\{n\}}^{(i)} \\ g_{\{n+1\}}^{(i)} \end{pmatrix} = \begin{pmatrix} g_{\{n,n+1\},1}^{(i)} \\ * \\ g_{\{n,n+1\},3}^{(i)} \\ * \end{pmatrix}, \tag{3.11}$$

Equivalently, let us define $D_{\{n,n+1\}}^{(i)}$ as

$$D_{\{n,n+1\}}^{(i)} = \begin{pmatrix} D_{\{n\}}^{(i)} & U_{\frac{nN}{N_{CN}^{(i)}}}^{(i)} \\ L_{\frac{nN}{N_{CN}^{(i)}}} & D_{\{n+1\}}^{(i)} \end{pmatrix} \tag{3.12}$$

.

(The element that are being computed will be written using **bold** characters, and the necessary ones for that computation are <u>underlined</u>.)

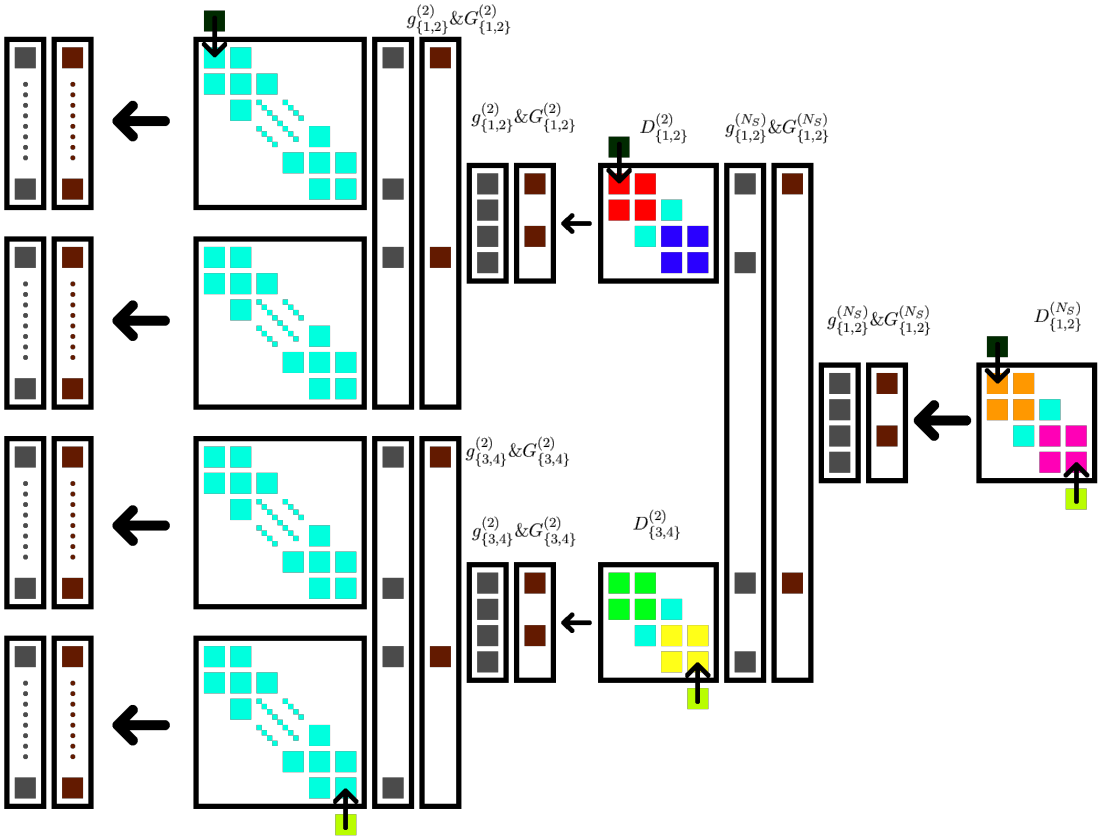

Figure 3.3: Reconstruction with $N_{CN} = 4$

lower node, $n+1$ The first element that can be computed on the lower node $n+1$ is $g^{(i)}_{\{n,n+1\},3}$. As recalled from the chapter on the Green's Function (2.2) it is possible to compute the first element of $g_{\{\tilde{n}\}}$ using $g_{\{\tilde{n}+1\}}$ with the formula 2.25. In this present case, $g_{\{\tilde{n}+1\},1}$ is $g^{(i)}_{\{n+2,1\}}$ which can be retrieved from stage $i+1$ as $g^{(i+1)}_{\{n+1\},1}$ (using 2.22)

$$g^{(i)}_{\{n,n+1\}} = \begin{pmatrix} * \\ * \\ \boldsymbol{g^{(i)}_{\{n,n+1\},3}} \\ * \end{pmatrix}, \tag{3.13}$$

$$D^{(i)}_{\{n,n+1\}} = \begin{pmatrix} d^{(i)}_{\{n,n+1\},1} & u^{(i)}_{\{n,n+1\},1} & 0 & 0 \\ l^{(i)}_{\{n,n+1\},1} & d^{(i)}_{\{n,n+1\},2} & u^{(i)}_{\{n,n+1\},2} & 0 \\ 0 & l^{(i)}_{\{n,n+1\},2} & \underline{d^{(i)}_{\{n,n+1\},3}} & \underline{u^{(i)}_{\{n,n+1\},3}} \\ 0 & 0 & \underline{l^{(i)}_{\{n,n+1\},3}} & \underline{d^{(i)}_{\{n,n+1\},4}} \end{pmatrix}, \tag{3.14}$$

$$g^{(i+1)}_{\{n+1\}} = \begin{pmatrix} \underline{g^{(i+1)}_{\{n+1\},1}} \\ * \end{pmatrix} \tag{3.15}$$

$$M \leftarrow \begin{pmatrix} \underline{d^{(i)}_{\{n,n+1\},3}} & \underline{u^{(i)}_{\{n,n+1\},3}} \\ \underline{l^{(i)}_{\{n,n+1\},3}} & \underline{d^{(i)}_{\{n,n+1\},4}} - u_{\frac{nN}{N^{(i)}_{CN}}} \underline{g^{(i+1)}_{\{n+1\},1}} l_{\frac{nN}{N^{(i)}_{CN}}} \end{pmatrix}^{-1} \tag{3.16}$$

$$\boldsymbol{g^{(i)}_{\{n,n+1\},3}} \leftarrow M_{1,1} \tag{3.17}$$

To speed up the computation, it is possible to take benefit from the fact that for a 2×2 matrix

$$B = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} \tag{3.18}$$

$$B_{1,1} = a^{-1} + a^{-1}b(d - ca^{-1}b)^{-1}ca^{-1} \tag{3.19}$$

which takes significantly less time than inverting the whole matrix. In the case of the pair $\{N^{(i)}_{CN} - 1, N^{(i)}_{CN}\}$ the matrix M becomes

$$M^{\Sigma(R)} \leftarrow \begin{pmatrix} \underline{d^{(i)}_{\{n,n+1\},3}} & \underline{u^{(i)}_{\{n,n+1\},3}} \\ \underline{l^{(i)}_{\{n,n+1\},3}} & \underline{d^{(i)}_{\{n,n+1\},4}} - \Sigma^{R(R)}_{bound} \end{pmatrix}^{-1} \tag{3.20}$$

and the result

$$\boldsymbol{g^{(i)}_{\{n,n+1\},3}} \leftarrow M^{\Sigma(R)}_{1,1} \tag{3.21}$$

upper node, n According to (2.22) again, we do not need to compute $g^{(i)}_{\{n,n+1\},1}$, only to retrieve it from the stage $i+1$

$$g^{(i)}_{\{n,n+1\}} = \begin{pmatrix} \boldsymbol{g^{(i)}_{\{n,n+1\},1}} \\ * \\ g^{(i)}_{\{n,n+1\},3} \\ * \end{pmatrix}, g^{(i+1)}_{\{n\}} = \begin{pmatrix} g^{(i+1)}_{\{n\},1} \\ \hline * \end{pmatrix} \tag{3.22}$$

$$\boldsymbol{g^{(i)}_{\{n,n+1\},1}} \leftarrow \underline{g^{(i+1)}_{\{n\},1}} \tag{3.23}$$

$$\tag{3.24}$$

just as $G^{(i)}_{\{n,n+1\},1}$ that we do not need to compute and retrieve from stage $i+1$ using (2.23).

$$G^{(i)}_{\{n,n+1\}} = \begin{pmatrix} \boldsymbol{G^{(i)}_{\{n,n+1\},1}} \\ * \\ * \\ * \end{pmatrix}, G^{(i+1)}_{\{n\}} = \begin{pmatrix} G^{(i+1)}_{\{n\},1} \\ \hline G^{(i+1)}_{\{n\}2} \end{pmatrix} \tag{3.25}$$

$$\boldsymbol{G^{(i)}_{\{n,n+1\},1}} \leftarrow \underline{G^{(i+1)}_{\{n\},1}} \tag{3.26}$$

For $G^{(i)}_{\{n,n+1\},2}$, it is possible to use (2.29), that allows to compute $G_{\{\tilde{n}\},2}$ using $G_{\{\tilde{n}-1\},2}$ and $g_{\{\tilde{n}+1\},1}$. In this case, $G_{\{\tilde{n}\},2} = G^{(i)}_{\{n,n+1\},2}$, $G_{\{\tilde{n}-1\},2} = G^{(i)}_{\{n-1\},2}$, and $g_{\{\tilde{n}+1\},1} = g^{(i)}_{\{n+1\},1}$. Notice that $G^{(i)}_{\{n-1\},2}$ and $g^{(i)}_{\{n+1\},1}$ can again be retrieved from the stage $i+1$.

$$G^{(i)}_{\{n,n+1\}} = \begin{pmatrix} G^{(i)}_{\{n,n+1\},1} \\ \boldsymbol{G^{(i)}_{\{n,n+1\},2}} \\ * \\ * \end{pmatrix}, \tag{3.27}$$

$$D^{(i)}_{\{n,n+1\}} = \begin{pmatrix} \underline{d^{(i)}_{\{n,n+1\},1}} & \underline{u^{(i)}_{\{n,n+1\},1}} & 0 & 0 \\ \underline{l^{(i)}_{\{n,n+1\},1}} & \underline{d^{(i)}_{\{n,n+1\},2}} & \underline{u^{(i)}_{\{n,n+1\},2}} & 0 \\ 0 & \underline{l^{(i)}_{\{n,n+1\},2}} & d^{(i)}_{\{n,n+1\},3} & u^{(i)}_{\{n,n+1\},3} \\ 0 & 0 & l^{(i)}_{\{n,n+1\},3} & d^{(i)}_{\{n,n+1\},4} \end{pmatrix}, \tag{3.28}$$

$$g^{(i)}_{\{n,n+1\}} = \begin{pmatrix} g^{(i)}_{\{n,n+1\},1} \\ * \\ \underline{g^{(i)}_{\{n,n+1\},3}} \\ * \end{pmatrix}, G^{(i+1)}_{\{n-1\}} = \begin{pmatrix} G^{(i+1)}_{\{n-1\},1} \\ \underline{G^{(i+1)}_{\{n-1\},2}} \end{pmatrix} \tag{3.29}$$

$$M \leftarrow \begin{pmatrix} \underline{d^{(i)}_{\{n,n+1\},1}} & \underline{u^{(i)}_{\{n,n+1\},1}} \\ \underline{l^{(i)}_{\{n,n+1\},1}} & \underline{d^{(i)}_{\{n,n+1\},2}} - u^{(i)}_{\{n,n+1\},2}\underline{g^{(i)}_{\{n,n+1\},3}}l^{(i)}_{\{n,n+1\},2} \end{pmatrix}^{-1} \tag{3.30}$$

$$\boldsymbol{G^{(i)}_{\{n,n+1\},2}} \leftarrow M_{2,2} + M_{2,1}u_{\frac{(n-1)N}{N^{(i)}_{CN}}}\underline{G^{(i+1)}_{\{n-1\},2}}l_{\frac{(n-1)N}{N^{(i)}_{CN}}}M_{1,2} \tag{3.31}$$

The case of the pair $\{1,2\}$ needs special attention. The formula to be used here is (2.31), and the result is then

$$M^{\Sigma(L)} \leftarrow \begin{pmatrix} \underline{d^{(i)}_{\{n,n+1\},1}} - \Sigma^{R(L)}_{bound} & \underline{u^{(i)}_{\{n,n+1\},1}} \\ \underline{l^{(i)}_{\{n,n+1\},1}} & \underline{d^{(i)}_{\{n,n+1\},2}} - u_{\frac{N}{N^{(i)}_{CN}}}\underline{g^{(i)}_{\{n,n+1\},3}}l_{\frac{N}{N^{(i)}_{CN}}} \end{pmatrix}^{-1} \tag{3.32}$$

$$\boldsymbol{G^{(i)}_{\{n,n+1\},2}} \leftarrow M^{\Sigma(L)}_{2,2} \tag{3.33}$$

lower node, $n+1$ The element $G^{(i)}_{\{n,n+1\},3}$ can easily be computed with the formula (2.17), the recursion of the general Green's Function. This can be done because the blocks $G^{(i)}_{\{n,n+1\},2}$ and $G^{(i)}_{\{n,n+1\},3}$ will be contiguous until stage 1.

$$G^{(i)}_{\{n,n+1\}} = \begin{pmatrix} G^{(i)}_{\{n,n+1\},1} \\ G^{(i)}_{\{n,n+1\},2} \\ \hline \boldsymbol{G^{(i)}_{\{n,n+1\},3}} \\ * \end{pmatrix}, g^{(i)}_{\{n,n+1\}} = \begin{pmatrix} g^{(i)}_{\{n,n+1\},1} \\ * \\ \underline{g^{(i)}_{\{n,n+1\},3}} \\ * \end{pmatrix} \tag{3.34}$$

$$\boldsymbol{G^{(i)}_{\{n,n+1\},3}} \leftarrow \underline{g^{(i)}_{\{n,n+1\},3}} + \underline{g^{(i)}_{\{n,n+1\},3}} u_{\frac{nN}{N^{(i)}_{CN}}} \underline{G^{(i)}_{\{n,n+1\},2}} l_{\frac{nN}{N^{(i)}_{CN}}} \underline{g^{(i)}_{\{n,n+1\},3}} \tag{3.35}$$

The last element $G^{(i)}_{\{n,n+1\},4}$ is easily retrieved from stage $i+1$ using (2.23).

$$G^{(i)}_{\{n,n+1\}} = \begin{pmatrix} G^{(i)}_{\{n,n+1\},1} \\ G^{(i)}_{\{n,n+1\},2} \\ G^{(i)}_{\{n,n+1\},3} \\ \boldsymbol{G^{(i)}_{\{n,n+1\},4}} \end{pmatrix}, G^{(i+1)}_{G_{\{n\}}} = \begin{pmatrix} G^{(i+1)}_{\{n\},1} \\ \underline{G^{(i+1)}_{\{n\},2}} \end{pmatrix} \tag{3.36}$$

$$\boldsymbol{G^{(i)}_{\{n,n+1\},4}} \leftarrow \underline{G^{(i+1)}_{\{n\},2}} \tag{3.37}$$

From stage 2 to stage 1 During the last step of the reconstruction, each node n can send its element $g^{(2)}_{\{n\},1}$ to the node $n-1$ for the latter node to compute its own $g^{(2)}_{\{n\},\frac{N}{N_{CN}}}$ using (2.15), the formula for contiguous elements. From there, each node is left with the information it needs to perform a Green's Function recursion using the formula (2.15) first for computing all the g elements. Then, using those newly computed g elements and the formula (2.17), it is possible to compute the whole G.

Nodes Now, let us have a look at a way of distributing the nodes for the reductions, from stage 2 on. Let $\mu^{(i)}$ hold the indices of the active nodes at the i^{th} reduction. The second node of each pair will store the blocks resulting from the reduction step, and thus the first node will stop working until the reconstruction phase.

$$\mu^{(1)} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ \vdots \end{pmatrix}, \mu^{(2)} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \\ 12 \\ \vdots \end{pmatrix} \mu^{(3)} = \begin{pmatrix} 4 \\ 8 \\ 12 \\ 16 \\ \vdots \end{pmatrix}, ..., \mu^{(N_R)} = \begin{pmatrix} \frac{N_{CN}}{2} \\ N_{CN} \end{pmatrix} \tag{3.38}$$

That way, $\mu_n^{(i)} = \mu_{\frac{n}{2}}^{(i+1)}$ for n even, and the node n at stage i is refered to as $\mu_n^{(i)}$. A few of the above formulas for computing G and g elements can then be simplified, as some of the necessary data is already stored in the node that will use it. Operations like (3.37) will not even be necessary anymore, if the blocks are sorted in a proper way.

3.1.3 Complexity

Let us now engage the complexity problem. At the first reduction step, N_{CN} nodes are working in parallel. At the second step, $N_{CN}^{(2)} = \frac{N_{CN}}{2}$. More generally

$$N_{CN}^{(i+1)} = \frac{N_{CN}^{(i)}}{2} \tag{3.39}$$

where $N_{CN}^{(i)}$ is the number of computing nodes working in parallel during the i^{th} reduction. We can then write

$$N_{CN}^{(i)} = \frac{N_{CN}}{2^{i-1}} \tag{3.40}$$

According to our definitions, the reduction $\tilde{N_S} - 1 = N_S$ (the last one before the system is completely reduced, see 3.1.1) will be performed with a single computing unit, and thus

$$N_{CN}^{(N_S)} = \frac{N_{CN}}{2^{N_S - 1}} = 1 \Leftrightarrow N_S = \log_2 (N_{CN}) + 1 \tag{3.41}$$

.

The computation time with respect to the number of computing nodes N_{CN} will come from four different operations types, executed in this order :

- one reduction step, with a system of size proportional to $\frac{N}{N_{CN}}$ (from stage 1 to stage 2) $\sim \frac{1}{N_{CN}}$

- $N_R - 1$ reduction steps, with systems of a constant size (from stage 2 to stage N_S) $\sim \log_2(N_{CN})$
- $N_R - 1$ reconstruction steps, with systems of a constant size (from stage N_S back to stage 2) $\sim \log_2(N_{CN})$
- one final reconstruction step, with a system of size proportional to $\frac{N}{N_{CN}}$ (from stage 2 to stage 1) $\sim \frac{1}{N_{CN}}$

Adding the complexity orders of each operation type, it yields $O(\frac{1}{N_{CN}} + \log_2 N_{CN})$. The graph of the function is presented here :

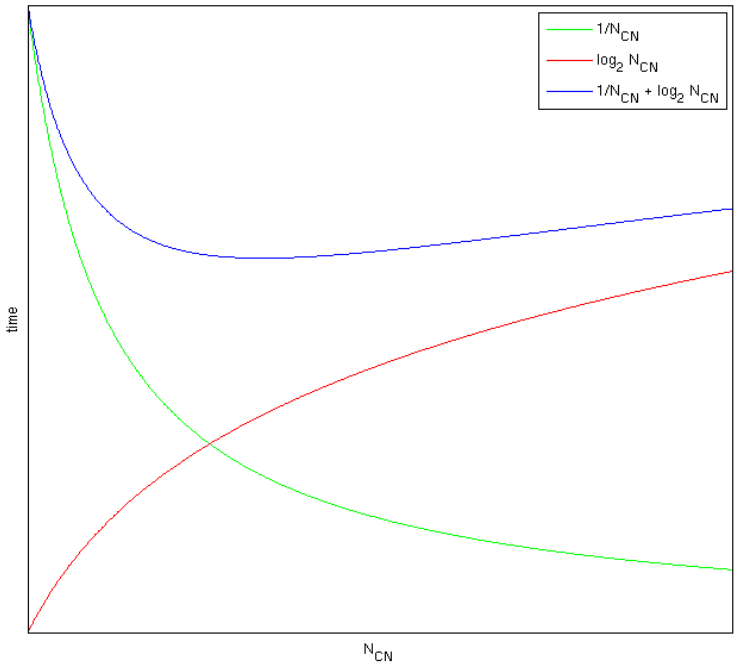

Figure 3.4: Computational time

With the number of computing nodes increasing the computational time will decrease until a certain point, but then rise again with a slope similar to the one of $\log_2 N_{CN}$, as the effect of $\frac{1}{N_{CN}}$ gets less and less strong. That means that the algorithm will run faster up to a certain number of computing nodes, but then, as the number of necessary stages increases, will slow down again.

3.2 Storage

Now the problem of storing the results will be addressed. Two different types of storage will be investigated : first the shared memory, where every node has access to everything that has been stored; subsequently, a separate memory scheme, each node having its own. The choice of whether using a shared or separate memory will ultimately depend on what is at our disposition

To be able to reconstruct the solution, the equivalent system after each reduction step will be needed.

3.2.1 Shared Memory

Each reduction step yields four non zero blocks per node. That is, if $N_d^{(i)}$ is the number of diagonal blocks created at reduction i, and $N_u^{(i)}$, $N_l^{(i)}$ the number of upper and lower blocks, respectively,

$$N_d^{(i)} = 2N_{CN}^{(i)} \tag{3.42}$$

$$N_u^{(i)} = N_{CN}^{(i)} \tag{3.43}$$

$$N_l^{(i)} = N_{CN}^{(i)} \tag{3.44}$$

.

The results of every reduction step have to be stored, and therefore $N_{d,u,l}$, the total number of elements created throughout a program run needs to be known for allocating memory. That will be

$$N_{d,u,l} = \Sigma_{i=1}^{N_R} N_{d,u,l}^{(i)} \tag{3.45}$$

And, using equation (3.39),

$$N_{CN}^{(i+1)} = \frac{N_{CN}^{(i)}}{2} = \frac{N_{CN}^{(i-1)}}{2 \cdot 2} = ... \Rightarrow N_{CN}^{(N_R - j)} = 2^j N_{CN}^{(N_R)} \tag{3.46}$$

Replacing i in 3.45 by $N_R - j$, (here for N_d)

$$N_d = 2\Sigma_{j=0}^{N_R-1}(2^j N_{CN}^{(N_R)}) = 2N_{CN}^{(N_R)}\Sigma_{j=0}^{N_R-1}2^j \tag{3.47}$$

and thus

$$N_d = 2 \cdot 2(2^{N_R} - 1) \tag{3.48}$$

$$N_{u,l} = 2(2^{N_R} - 1) \tag{3.49}$$

$$N_B = N_d + N_u + N_l = 8(2^{N_R} - 1) \tag{3.50}$$

or

$$N_B = 8(N_{CN} - 1) \tag{3.51}$$

defining N_B, the total number of extra blocks saved during the program run.

Implementation To find out how many blocks are still to be saved after a reduction i one can note that it is actually the same problem with a first number of computing nodes $\tilde{N}_{CN} = \frac{N_{CN}}{2^{i-1}}$. It means that the number of blocks that have already been stored at a stage $j = i+1$ (after the reduction i) can be computed :

$$N_B - \tilde{N}_B = 8(N_{CN} - 1) - 8(\tilde{N}_{CN} - 1) = 8(N_{CN}(1 - \frac{1}{2^{j-2}})) \tag{3.52}$$

If memory was previously allocated with the size N_B, the result 3.52 can then be used as an offset to store and fetch the blocks that are relevant at a given stage. That subsystem will be of size $4\frac{N_{CN}}{2^{j-2}}$ blocks.

3.2.2 Separate Memories

If the memories are separate, it is possible to first allocate some memory in a size proportional to N_B. Another approach is to first compute the number of blocks that have to be stored, and then allocate a minimum amount of memory. In that case, we will be interested in $N_{B,n}$, the number of extra blocks stored for the node of index n. At first we will determine the number of reductions a computing node will take part in, $N_{R,n}$. Once this is done, the result will be very straight forward.

At each reduction, only the nodes whose position is even in $\mu^{(i)}$ will hold results and "survive" (see paragraph *Nodes*, 3.1.2). Since only every second index will be kept, if a node was at the index a for the stage i it will be moved to $\frac{a}{2}$ for stage $i+1$. At each stage the index is then divided by two, and if the node will take part in the next reduction will depend on if new index is even or not (more formally in the appendix B). Then, if $N_{R,i}$ is the number of states during which the node will have to store results,

$$N_{B,p} = 4N_{R,p} \tag{3.53}$$

with p the node.

Implementation For this memory scheme, the offset is as simple as $4(i-2)$, with i the stage.

Chapter 4

Tests

During the course of the project, a Matlab script was implemented. Unfortunately, the Matlab version did not provide parallelism, and so the behavior had to be simulated. A hundred different (random) systems have been simulated and the time of simulation measured. Each of the system has been solved with 2^2 to 2^{10} computing nodes, and the final results have been averaged over all the simulations and plotted. Blocks were replaced with single entries.

During the reduction, the runtime was measured for each virtual node, and then the maximum was taken. During the reconstruction, the runtime was also measured for each virtual node pair, and then the maximum was taken. For both the reduction and the reconstruction, the computational time is the summation of each step's maximum. It can be noticed that the computational times of the first reduction's step (from stage 1 to stage 2) and of the last reconstruction's step (from stage 2 to stage 1) have a very different behavior than the rest of the reduction and reconstruction steps. As the latter ones operate as $\sim O(log_2 N_{CN})$, the steps between stages 1 and 2 and between 2 and 1 operate as $\sim O(\frac{1}{N_{CN}})$.

Figure 4.1 shows the measurements. Very soon, the logarithmic behavior of the reduction and reconstruction takes over the effect of the fully parallel operations. Figure 4.2 presents the same results with also a linear scale on the x-axis but with a closer look at the origin. Unfortunately, in our case, it shows that the minimum of computational time is reached with eight computing nodes; adding more nodes in parallel will not decrease the total time at all, quite the opposite, it will increase it in a logarithmic fashion.

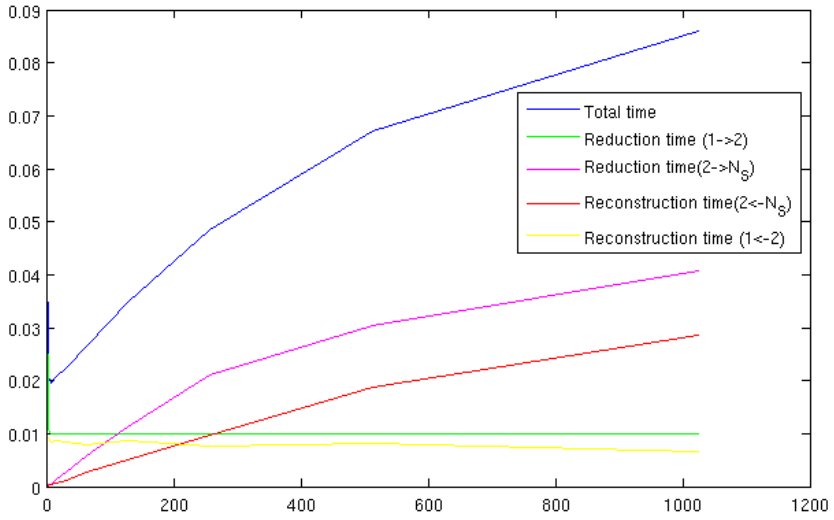

Figure 4.1: Computational time, function of N_{CN}

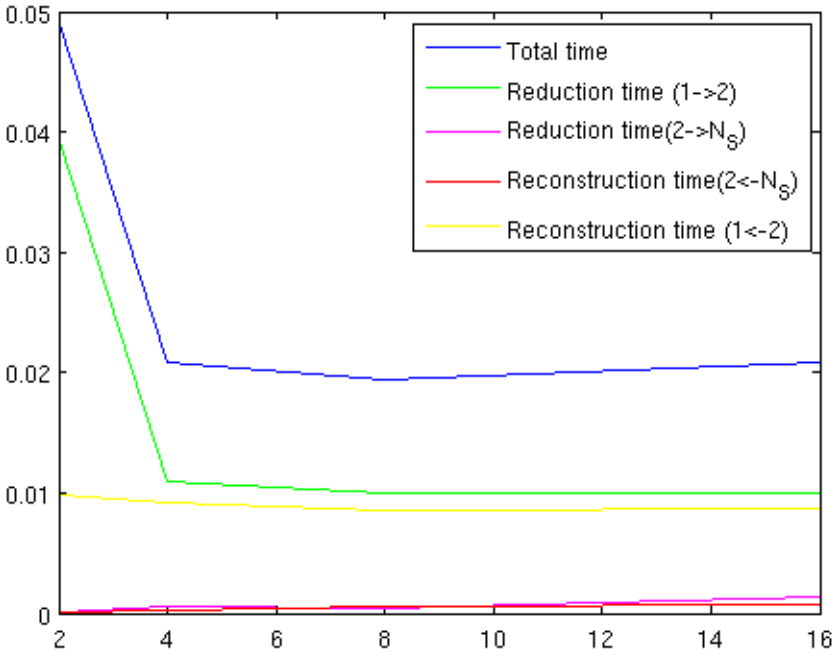

Figure 4.2: Computational time, function of N_{CN}, origin

Figure 4.3 presents the result in a more condensed way with a logarithmic scale for N_{CN}. It is interesting to notice that the slope of both the reduction ($2 \rightarrow N_S$) and the reconstruction ($N_S \leftarrow 2$) are not constant from the beginning. That means that their computational time does not increase as a logarithm from the beginning, but a bit slower until approximately 128 computing nodes for the reconstruction and 64 for the reduction. The hypothesis is that the computational time is below Matlab's precision limit for $N_{CN} \leq 64$ and 128 nodes, and thus a time equivalent to 0 is measured.

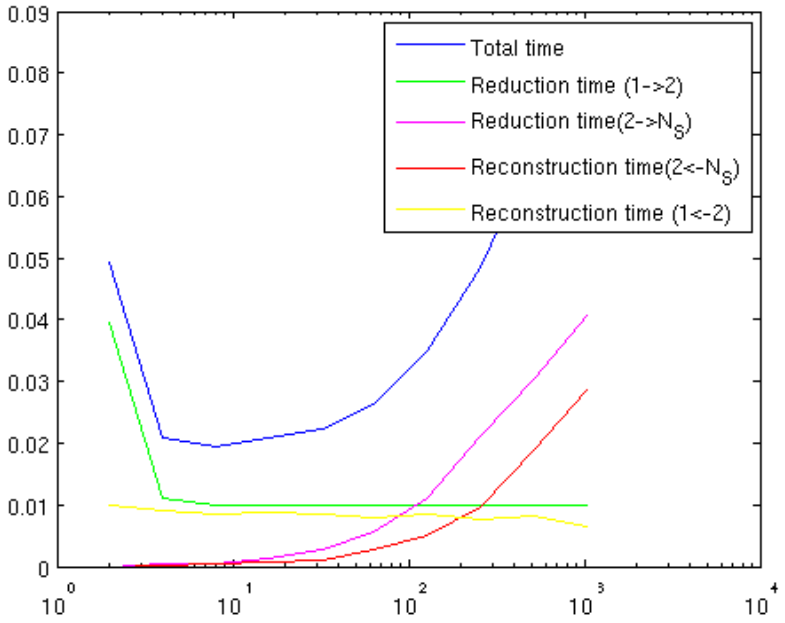

Figure 4.3: Computational time, function of N_{CN}, logarithmic scale for N_{CN}

Both the reduction ($N_S \rightarrow 2$) and the reconstruction ($2 \leftarrow N_S$) seem not to be affected by N_{CN} for $N_{CN} = 8$ and more, although their computational time should keep decreasing, according to what was discussed in section 3.1.3 (as the N_{CN} gets bigger, the systems the first step has to deal with decrease in size as $\frac{N}{N_{CN}}$). This might be due to Matlab reaching its precision limit for measuring time, or even due to operations that are not computational (memory load,...).

Chapter 5

Outlook

The next step is to implement the algorithm on a system capable of real parallelism. It can then be interesting to study the effect of a bigger N (the number of blocks in the system's matrix diagonal) on the algorithm's efficiency. Since only the computing part with complexity $\sim \frac{N}{N_{CN}}$ depends on N, and as with greater N its curve is shifted to the right, the overall time function should see its minimum shifted to the right as well. Note that the blocks size should not change the time function's shape, only move it up or down.

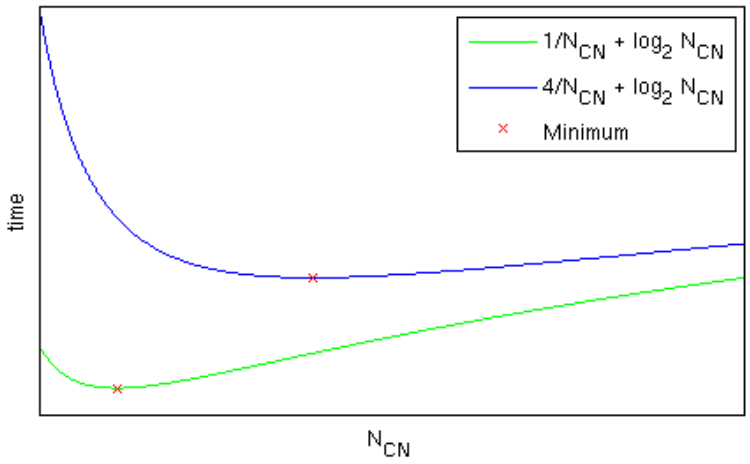

Figure 5.1: Effect of N on the minimum's position

Further research on the topic of the parallelisation of the Green's Function could involve avoiding any kind of sequential part. Instead of having a pair of computing nodes $\{n, n+1\}$ working sequentially during the reconstruction, one could look into a method for solving the problem using a Green's Function taking only $\Sigma^{R(L)}_{bound}$ into account on the node n, and a

Green's Function taking only $\Sigma_{bound}^{R(R)}$ into account on the node $n+1$. Both nodes could complete their function independently, n in a top-down way, $n+1$ in a bottom up way. Once this is done, the nodes could somehow exchange the information they hold on their systems, and each node could finally compute the real Green's Function G^R taking both boundary energies into account. This method should be twice as fast as the one implementend in this paper.

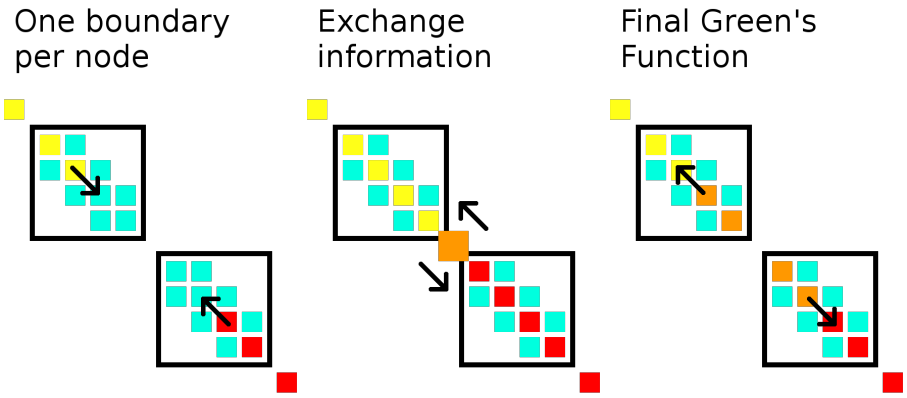

Figure 5.2: New method

Appendix A

Notations

Name	Description
d_k,u_k,l_k	The k^{th} block element of the diagonal, the upper diagonal and lower diagonal of a matrix, respectively
$d_{\{n\},k}$	The k^{th} block element of the diagonal on the node n
N	The size of the system's matrix diagonal (in blocks)
E	The diagonal matrix whose elements are the energy level we are trying to simulate
H	The Hamiltonian of the system
Σ^R_{bound}	The boundary self-energy matrix
Σ^R_{scatt}	The scattering self-energy matrix
D	The system $E - H - \Sigma^R_{scatt}$
G^R	The Retarded Green's Function
g^R	The first stage of the Retarded Green's Function
$D_{\{n\}}$	The submatrix of D on the n^{th} computing node
$D_{\{n,n+1\}}$	The submatrix of D on the pair of computing nodes $\frac{n+1}{2}$
$g^R_{\{n\}}, G^R_{\{n\}}$	The elements of g^R and G^R corresponding to the system $D_{\{n\}}$
$g^R_{\{n,n+1\}}, G^R_{\{n,n+1\}}$	The elements of g^R and G^R corresponding to the system $D_{\{n,n+1\}}$
$D^{(i)}$	The system D reduced at a stage i
$g^{(i)}, G^{(i)}$	g^R and G^R for a reduced system $D^{(i)}$
N_{CN}	The number of computing nodes availabe for the parallelisation
$N^{(i)}_{CN}$	The number of nodes used at the reduction step i
$\mu^{(i)}$	The vector containing the indices of the computing nodes used at stage i
$\tilde{N}_R$	The number of reduction steps necessary to reduce the original system down to a 2×2 matrix
$\tilde{N}_S$	The number of stages involved in a reduction from the original system down to a 2×2 matrix

Name	Description
N_R	The number of reduction steps necessary to reduce the original system down to a 4×4 matrix
N_S	The number of stages involved in a reduction from the original system down to a 4×4 matrix
N_d, N_u, N_l	The number of diagonal, upper diagonal and lower diagonal elements created during the reduction, respectively
$N_d^{(i)}, N_u^{(i)}, N_l^{(i)}$	The number of diagonal, upper diagonal and lower diagonal elements created during the reduction step i, respectively
N_B	The total number of new blocks created and stored during the reduction phase
$N_{S,n}$	The number of reduction steps a computing n will store the resulting systems for
$N_{R,n}$	The number of reduction steps a computing n will take part in
$N_{B,n}$	The total number of new blocks created and stored during the reduction phase on the node n

Appendix B

The Extended Parity Algorithm

This algorithm determines how many times a number can be divided by two until the result is not even anymore.

Algorithm 1 Extended Parity Algorithm

```
1: procedure XtdParity(n)
2:     a ← n
3:     s ← 1
4:     while a is even do
5:         a ← b
6:         s ← s + 1
7:     end while
8:     return s
9: end procedure
```