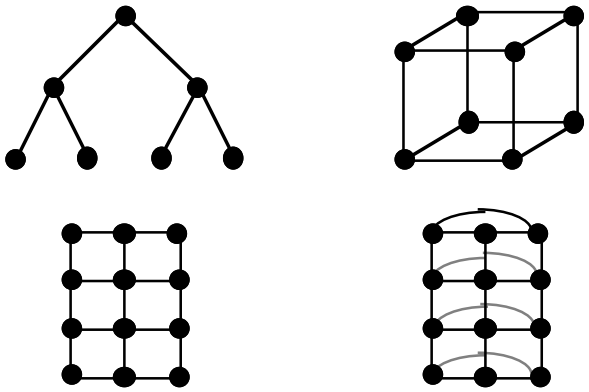
Time complexity of a parallel conjugate gradient solver for light scattering simulations: theory and SPMD implementation

Sloot, P.M.A.; Hoekstra, A.G.; Hoffmann, W.; Hertzberger, L.O.

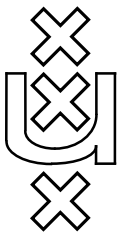
Time Complexity of a Parallel Conjugate Gradient Solver for Light Scattering Simulations:
Theory and SPMD Implementation

A.G. Hoekstra, P.M.A. Sloot, W. Hoffmann and L.O. Hertzberger

Parallel Scientific Computing Group, Department of Computer Systems
Faculty of Mathematics and Computer Science
University of Amsterdam



Technical Report



Time Complexity of a Parallel Conjugate Gradient Solver for Light Scattering Simulations:
Theory and SPMD Implementation

ABSTRACT

We describe parallelization for distributed memory computers of a preconditioned Conjugate Gradient method, applied to solve systems of equations emerging from Elastic Light Scattering simulations. The execution time of the Conjugate Gradient method is analyzed theoretically. First expressions for the execution time for three different data decompositions are derived. Next two processor network topologies are taken into account and the theoretical execution times are further specified as a function of these topologies. The Conjugate Gradient method was implemented with a rowblock data decomposition on a ring of transputers. The measured - and theoretically calculated execution times agree within 5 %. Finally convergence properties of the algorithm are investigated and the suitability of a polynomial preconditioner is examined.

KEYWORDS: Elastic Light Scattering, preconditioned conjugate gradient method, data decomposition, time complexity analysis, performance measurement, transputer network

June 1992

Technical Report

Parallel Scientific Computing Group
Department of Computer Systems
Faculty of Mathematics and Computer Science
University of Amsterdam

progress report of the FOM Fysische Informatica project "Simulation of Elastic Light Scattering from micron sized Particles by means of Computational Physics Methods", grant number NWO 810-410-04 1.

1] INTRODUCTION

Elastic Light Scattering (ELS) is a powerful non destructive particle detection and recognition technique, with important applications in diverse fields such as astrophysics, biophysics, or environmental studies. For instance, light scattered from (nucleated) blood cells contains information on both the cell size and the morphology of the cell [1,2,3,4,5]. The number of theories describing light scattering from biological cells is limited [6,7]. Moreover, no adequate general theory is known yet to describe analytically the ELS characteristics of arbitrary shaped particles (although the need for such a theory is generally appreciated).

Many exact and approximate theories to calculate ELS from particles are known [8]. Nevertheless, important classes of particles fall outside the range of these theories. This started much research in the field of light scattering by arbitrary shaped particles [9]. The coupled dipole method, due to Purcell and Pennypacker [10], is one method that in principle allows calculation of ELS from any particle.

In September 1990 the project entitled "Simulation of Light Scattering from Micron Sized Particles by means of Computational Physics Methods" started in the department of Computer Systems of the Faculty of Mathematics and Computer Science of the University of Amsterdam. It is the purpose of this project to show that the coupled dipole method can be applied to simulate the ELS of **all** types of small particles and that this is of fundamental interest in e.g. the development of cell characterization techniques. Recent progress in large vector-processing super-computers and parallel-processing systems on the one side, and advanced optical detection equipment on the other hand, will allow us to investigate in detail the light scattering properties of, for instance, human white blood cells.

The computational most demanding part of the coupled dipole method is a large set of linear equations that must be solved. Particles of our interest, human white bloodcells, give rise to matrices with dimensions of $O(10^4)$ to $O(10^6)$. To keep calculation times within acceptable limits, a very efficient solver, implemented on a powerful (super)computer is required. We apply a Conjugate Gradient (CG) method, implemented on a transputer network, to solve the system of equations.

This technical report concentrates on the functional and implementation aspects of parallelizing a CG method suited to our application. After a description of ELS and the coupled dipole method in section 2, section 3 gives a theoretical time complexity analysis of the CG method for different parallelization strategies. Based on the results of section 3 the CG method was implemented on a bi-directional ring of transputers, with a rowblock decomposition of the system matrix. Section 4 describes this implementation, and section 5 presents performance measurements and convergence behaviour of the method. The results are discussed in section 6, and finally conclusions are drawn in section 7.

2] ELASTIC LIGHT SCATTERING FROM SMALL PARTICLES

2.1 Introduction

Consider a particle in an external electromagnetic field. The applied field induces an internal field in the particle and a field scattered from the particle. The intensity of the scattered field in the full solid angle around the particle can be measured. This ELS pattern can be viewed as a fingerprint of the particle and is extremely sensitive to properties of the particle [8]. Therefore it is possible to distinguish different particles by means of ELS. This non-destructive

remote sensing of particles is a very important application of ELS. The question arises wether it is possible to fully describe a particle solely on the basis of its complete ELS pattern (the inverse scattering problem). In principle this is impossible without knowledge of the internal field in the particle [11]. Fortunately, in most application we have some idea of the particles involved and in that case it usually is possible to solve the inverse problem within a desired accuracy. In most cases measurement of just a very small part of the ELS pattern suffices to be able to identify several particles.

To solve the inverse scattering problem, or to define small parts of the ELS pattern which are very sensitive to identify particles requires a theory to calculate the ELS of an arbitrary particle. Sloot et al.[1,12] give an overview of analytical and approximation theories for ELS, and conclude that for ELS from human white bloodcells no suitable theory exists to calculate the complete ELS pattern. Hage [13] draws the same conclusion for his particles of interest; interplanetary and interstellar dust particles. In general one can conclude that for most applications it is up till now not possible to calculate the ELS pattern. As a consequence, many research groups rely on the analytical Mie theory of scattering by a sphere (see for instance ref. [14]), although it is known that even small perturbations from the spherical form induce notable changes in the ELS pattern [15,16]. This prompted much research to scattering by arbitrary shaped particles [9]. We intend to calculate the ELS pattern by means of the so-called Coupled Dipole method [10].

2.2 Basic theory of ELS

We will introduce some basic definitions and notations to describe the ELS. The full details can be found in text books of e.g. Bohren and Huffman [8] or van der Hulst [7].

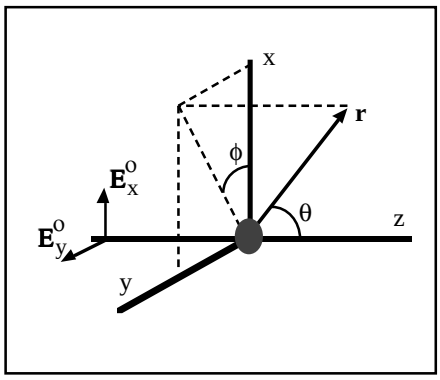


Figure 1: Scattering geometry

Figure 1 gives the basic scattering geometry. A particle is situated in the origin, and is illuminated by an incident beam travelling in the positive z-direction. A detector at **r** measures the intensity of the scattered light. The distance $|\mathbf{r}|$ is very large compared to the size of the particle. The far field scattered intensity is measured. The far field is only dependant on the angles θ and ϕ (and a trivial $1/|\mathbf{r}|$ dependence due to the spherical wave nature of the far field) [17]. The plane through **r** and the wave vector of the incident beam (in this case the z-axes) is called the scattering plane. The angle θ between the incident wave vector and **r** is the scattering angle. As a simplification we assume $\phi = \pi/2$, the yz plane is the scattering plane. The incident and scattered electric field are resolved in components perpendicular (subscript $\perp$) and parallel (subscript $||$) to the scattering plane. In this case $(\mathbf{E}^0)_{||} = (\mathbf{E}^0)_y$ and $(\mathbf{E}^0)_\perp = (\mathbf{E}^0)_x$, where the superscript 0 denotes the incident light. The formal relationship between the incident electric field and the scattered electric field (superscript s) is

$$[1] \qquad \begin{pmatrix} \mathbf{E}^s_{||} \\ \mathbf{E}^s_\perp \end{pmatrix} = \frac{e^{ik(r-z)}}{-ikr} \begin{pmatrix} S_2 & S_3 \\ S_4 & S_1 \end{pmatrix} \begin{pmatrix} \mathbf{E}^0_{||} \\ \mathbf{E}^0_\perp \end{pmatrix}.$$

The matrix elements S_j (j = 1,2,3,4) are the amplitude scattering functions, and depend in

general on θ and ϕ. Another convenient way to describe ELS is in the frame work of Stokes vectors and the Scattering matrix [8, chapter 3.3].

The scattering functions $S_j(\theta,\phi)$ depend on the shape, structure and optical properties of the particle. Analytical expressions only exist for homogeneous (concentric) spheres, (concentric) ellipsoids and (concentric) infinite cylinders. Furthermore, a lot of approximations for limiting values of the scattering parameter α* and the relative refractive index m exist (see [7], paragraph 10.1).

2.3 ELS from human leukocytes

In previous years we developed both theory and experimental equipment to study the ELS from biological particles [18,19,20,21,22,12]. Sloot developed an extension of the RDG scattering for concentric spheres (called mRDG) [19], which was very well suited to explain the anomalous Forward Scattering (FS) of osmotically stressed human lymphocytes [20]. Based upon mRDG calculations of the FS of omsotically stressed lymphocytes we even predicted a totally unexpected biological phenomenon, the change of nuclear volume of lymphocytes in aniosmotic conditions. Recently we have proven this prediction to be correct [21].

Although the simple analytical mRDG theory can explain certain ELS characteristics of leukocytes, we have shown in [12] and [23] that the ELS pattern of leukocytes contains polarization information, which cannot be described by mRDG. It is just this polarized ELS which facilitates differentiation between e.g. neutrophylic and eosinophylic granulocytes [24]. Recent experiments carried out by our group, in collaboration with Prof. Dr. J. Greve, University of Twente, the Netherlands, show that the scattering matrix of most human leukocytes deviate from the simple mRDG approximation [to be published].

Obviously a more strict and fundamental approach is required to calculate the details of the ELS pattern of human leukocytes. We cannot use the Mie theory, nor first order approximations as mRDG. The relevant parameter α and m for human leukocytes are in the range $10 \le \alpha \le 100$ and $1.01 \le m \le 1.1$§ [1]. Therefore other well known approximations (as anomalous diffraction or ray tracing) cannot be applied [7]. We have to rely on numerical techniques to integrate the Maxwell equations, or on discrete theories like the Coupled Dipole formulation of ELS.

2.3 The coupled dipole method

In the coupled dipole method of ELS a particle is divided into N small subvolumes called dipoles. Dipole i (i = 1,..,N) is located at position $\mathbf{r}_i$ ($\mathbf{r} = (x,y,z)^T$). An externally applied electric field $\mathbf{E}^0(\mathbf{r})$ is incident on the particle ($\mathbf{E}(\mathbf{r}) = (E_x(\mathbf{r}),E_y(\mathbf{r}),E_z(\mathbf{r}))^T$). An internal electric field $\mathbf{E}(\mathbf{r}_i)$ at the dipole sites, due to the external field and the induced dipole fields is generated. The scattered field at an observation point $\mathbf{r}_{obs}$ can be calculated by summing the electric fields radiated by all N dipoles. The scattered field $\mathbf{E}^s(\mathbf{r}_{obs})$ as measured by an observer at $\mathbf{r}_{obs}$ is

$$[2] \qquad \mathbf{E}^s(\mathbf{r}_{obs}) = \sum_{j=1}^{N} \mathbf{F}_{obs,j}\mathbf{E}(\mathbf{r}_j),$$

---

* The scattering parameter α is defined as $\alpha = 2\pi r/\lambda$, where λ is the wavelength of the incident light, and r a measure of the particle size.

§ We assume that the particles are suspended in water, and the wavelength of the incident light to be in the visible range (400 nm $\le \lambda \le$ 700 nm).

where $\mathbf{F}_{i,j}$ is a known 3×3 matrix of complex numbers describing the electric field at $\mathbf{r}_i$, radiated by a dipole located at $\mathbf{r}_j$. The field radiated by dipole j is calculated by multiplying $\mathbf{F}_{i,j}$ with the electric field $\mathbf{E}(\mathbf{r}_j)$ at dipole j. $\mathbf{F}_{i,j}$ depends on the relative refractive index of the particle, the wavelength of the incident light and the geometry of the dipole positions.

As soon as the internal field $\mathbf{E}(\mathbf{r}_i)$ is known, the scattered field can be calculated. The electric field at dipole i consists of two parts: the external electric field (i.e. $\mathbf{E}^0(\mathbf{r}_i)$ and the electric fields radiated by all other dipoles:

$$[3] \qquad \mathbf{E}(\mathbf{r}_i) = \mathbf{E}^0(\mathbf{r}_i) + \sum_{j \neq i}^{N} \mathbf{F}_{i,j}\mathbf{E}(\mathbf{r}_j) \quad .$$

Equation 3 results in a system of N coupled linear equations for the N unknown fields, which can be formulated as a matrix equation

$$[4] \qquad \mathbf{AE} = \mathbf{E}^0 \ ,$$

where

$$[5] \qquad \mathbf{E} = \begin{pmatrix} \mathbf{E}(\mathbf{r}_1) \\ \vdots \\ \mathbf{E}(\mathbf{r}_N) \end{pmatrix} \text{ and } \mathbf{A} = \begin{pmatrix} \mathbf{a}_{1,1} & \cdots & \mathbf{a}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{N,1} & \cdots & \mathbf{a}_{N,N} \end{pmatrix},$$

with $\mathbf{a}_{i,i} = \mathbf{I}$ (the 3×3 identity matrix) and $\mathbf{a}_{i,j} = -\mathbf{F}_{i,j}$ if $i \neq j$. The vector $\mathbf{E}^0$ has the same structure as **E**. The complex matrix **A** is referred to as the interaction matrix. Equation 4 is a set of 3N equations of 3N unknowns (the 3 arising from the 3 spatial dimensions). All numbers in equation 4 are complex. The 3N×3N interaction matrix is a dense symmetric matrix.

The size of the dipoles must be small compared to the wavelength of the incident light ($d \sim \lambda/10$, with d the diameter of a spherical dipole, and λ the wavelength of the incident light). We are interested in light scattering from particles with sizes in the order of 3 to 15 times the wavelength of the incident light. A crude calculation shows that in this case the number of dipoles N lies in the range $O(10^4)$ to $O(10^6)$. Therefore equation 4 is a very large linear system. Calculation of the internal electric fields at the dipole sites, that is to solve the system linear of equation 4 is the computational most demanding part of the coupled dipole method. In the sequel we will address this problem in detail.

2.4] Numerical considerations

From a numerical point of view, the coupled dipole method boils down to a very large system of linear equations $\mathbf{Ax} = \mathbf{b}$, with **A** an n×n complex symmetric matrix, **b** a known complex vector and **x** the unknown complex vector. Linear systems are solved by means of direct - or iterative methods [25]. Direct methods, such as LU factorization, require $O(n^3)$ floating-point operations to find a solution, and must keep the complete (factored) matrix in memory. On the other hand iterative methods, such as the Conjugate Gradient method, require $O(n^2)$ floating-point operations per iteration. The k-th iteration yields an approximation $\mathbf{x}_k$ of the wanted vector **x**. If $\mathbf{x}_k$ satisfies an accuracy criterion, the iteration has converged and $\mathbf{x}_k$ is accepted as the solution of the linear system. If the spectral radius and the condition number of the system matrix satisfy some special criteria, iterative methods will converge to the solution **x**

[25], and the total number of iterations q needed can be much smaller than n. The total number of floating-point operations to find a solution is $O(qn^2)$, which is much less than with direct methods. Furthermore, iterative methods do not require the system matrix being kept in memory.

Merely the size of the system matrix forces the use of iterative methods. Suppose one floating-point operation takes 1.0 μs, and $n = 3.0\ 10^5$. A direct method roughly needs O(850) years to find a solution. Even on a 1000 node parallel computer, assuming ideal linear speedup, a direct method still requires O(10) months. An iterative method needs O(25) hours per iteration, which is O(100) seconds on the ideal 1000 node parallel machine. If $q << n$, calculation times can be kept within acceptable limits, provided that the implementation can run at a sustained speed in the Gflop/s range.

2.3] The conjugate gradient method

A very powerful iterative method is the Conjugate Gradient (CG) method [25]. Usually this method is applied to systems with large sparse matrices, like the ones arising from discretizations of partial differential equations. Draine however showed that the CG method is also very suited to solve linear systems arising from the coupled dipole method [26]. For instance, for a typical particle with 2320 dipoles (n = 6960) the CG method only needs 17 iterations to converge.

The original CG method of Hestenes and Stiefel [27] (the CGHS method in Ashby's taxonomy [28], to which we will confirm ourselves) is only valid for Hermitian positive definite (hpd) matrices. Since in the coupled dipole method the matrix is not Hermitian (but symmetric), CGHS cannot be employed. We will use the PCGNR method, in the Orthomin implementation (see reference [28] for details). Basically this is CGHS applied to the normal equation

[6] $$\mathbf{A}^H \mathbf{A}\, \mathbf{x} = \mathbf{A}^H \mathbf{b},$$

(the superscript H denotes the Hermitian of the matrix), with total preconditioning. The PCGNR method is suitable for any system matrix [28]. The algorithm is shown below:

The PCGNR algorithm

Initialize: Choose a start vector $\mathbf{x}_0$ and put

$k = 0$

$\mathbf{r}_0 = (\mathbf{b} - \mathbf{A}\mathbf{x}_0)$ calculate the residual vector

$\mathbf{s}_0 = \mathbf{M}^{-1}\mathbf{M}^{-H}\mathbf{A}^H\mathbf{r}_0$

$\mathbf{p}_0 = \mathbf{s}_0$ the first direction vector

Iterate: **while** $|\mathbf{r}_k| \geq \varepsilon |\mathbf{b}|$ iterate until the norm of the the residual vector is small enough

$$\alpha_k = \frac{(\mathbf{A}^H\mathbf{r}_k)^H\mathbf{s}_k}{(\mathbf{A}\mathbf{p}_k)^H(\mathbf{A}\mathbf{p}_k)}$$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$ calculate new iterate

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k(\mathbf{A}\mathbf{p}_k)$ update residual vector

$\mathbf{s}_{k+1} = \mathbf{M}^{-1}\mathbf{M}^{-H}\mathbf{A}^H\mathbf{r}_{k+1}$

$$\beta_k = \frac{(\mathbf{A}^H \mathbf{r}_{k+1})^H \mathbf{s}_{k+1}}{(\mathbf{A}^H \mathbf{r}_k)^H \mathbf{s}_k}$$

$\mathbf{p}_{k+1} = \mathbf{s}_{k+1} + \beta_k \mathbf{p}_k$ calculate new direction vector

k = k + 1

stop $\mathbf{x}_k$ is the solution of $\mathbf{Ax} = \mathbf{b}$

The number ε is the stopping criterion, and is set to the squareroot of the machine precision. Since all implementations will be in double precision (i.e. a 52 bit fraction) ε is set to 10^{-8}. The vector $\mathbf{r}_k$ is the residual vector of the k-th iteration, the vector $\mathbf{p}_k$ is the direction vector, and **M** is the preconditioning matrix. The iterations stops if the norm of $\mathbf{r}_k$ is smaller than the norm of **b** multiplied by ε.

The purpose of preconditioning is to transform ill-conditioned system matrices to a well conditioned form, thus increasing the convergence rate of the conjugate gradient method. The preconditioning matrix **M** must approximate the system matrix **A** as closely as possible but still allow a relative easy calculation of the vector $\mathbf{s}_k$. A good preconditioner decreases the total execution time of the conjugate gradient process. This means that a good parallel preconditioner not only decreases the total number of floating-point operations, but also possesses a high degree of parallelism. A good preconditioner depends both on the system matrix and the parallel computer. For instance, the incomplete Cholesky factorization preconditioner [29] is very successful on sequential computers, but performs not as good on vector- and parallel computers.

Polynomial preconditioners [30] are very well suited for parallel computers [31, 32], and experiments have shown that, implemented on a distributed memory computer, they can be much more effective than incomplete factorization preconditioners [see e.g. 33]. Therefore we adapt the concept of polynomial preconditioning and put

[7] $$\mathbf{M}^{-1} = \sum_{i=0}^{m} \gamma_i \mathbf{A}^i \quad .$$

The choice of m and γ_i is topic of active research [e.g. 32,34], but is beyond the scope of our research. Here we concentrate on parallelization of the PCGNR method. We take the von Neumann series as the polynomial preconditioners.

[8] $$\mathbf{M}^{-1} = \sum_{i=0}^{m} \mathbf{N}^i ,$$

where **N** = **I** - **A**.

3] TIME COMPLEXITY ANALYSIS

3.1 Introduction and formal method

This section presents the functional aspects of parallelizing the PCGNR method for matrices coming from the coupled dipole method. Parallelism is realized by decomposition of

the problem. This can be a task decomposition or a data decomposition [35]. In the first case the problem is divided into a number of tasks that can be executed in parallel, in the second case the data is divided in groups (grains) and the work on these grains is performed in parallel. The parallel parts of the problem are assigned to processing elements. In most cases the parallel parts must exchange information. This implies that the processing elements are connected in a network. Parallelizing a problem basically implies answering two questions: what is the best decomposition for the problem and which processor interconnection scheme is best suited.

The PCGNR algorithm basically consists of a big while loop. The consecutive instances of the loop depend on previously calculated results, and therefore cannot be executed in parallel. The same is true for the tasks inside one instance of the loop. This means that task decomposition is not applicable for the PCGNR algorithm. Parallelism will be introduced by means of data decomposition.

A metric is needed to decide which decomposition and interconnection scheme should be selected. We choose the total execution time of the parallel program T_{par}. The decomposition and interconnection that minimize T_{par} must be selected. In case of data parallelism the execution time T_{par} depends on two variables and on system parameters:

[9] $$T_{par} \equiv T_{par}(p,n; \text{system parameters}) ;$$

p is the number of processing elements and n a measure of the data size.

The system parameters emerge from an abstract model of the parallel computing system. A computing system is defined as the total complex of hardware, system software and compiler for a particular high level programming language. A time complexity analysis requires a reliable, however not too complicated abstraction of the computing system.

At this point we will restrict ourselves. First of all we assume that the parallel computing system is an instance of Hoare's Communicating Sequential Processes [36]. This means that the system consists of a set of sequential processes, running in parallel and exchanging data by means of synchronous point to point communication. The abstract model needs to specify how the parallel processors† are connected (the network topology), the time needed to send a certain amount of data over a link connecting two processors, and execution times on a single processor. Our target platform, a Meiko computing surface consisting of 64 transputers, programmed in OCCAM2, clearly falls within this concept.

The communication is described by a startup time $\tau_{startup}$ plus a pure sending time $n\tau_{comm}$, where n is the number of bytes that is sent, and τ_{comm} the time to send one byte.

The next assumption is that the action within the sequential processes mainly consists of manipulating floating-point numbers (which is obviously true for numerical applications), and that the time needed to perform one floating-point operation can be described with a single parameter τ_{calc}. At this point T_{par} can be further specified:

[10] $$T_{par} \equiv T_{par}(p,n; \tau_{calc},\tau_{startup},\tau_{comm},\text{topology}).$$

Finally we assume that the parallel program consists of a sequence of *cycles* [37]. During a cycle all processors first perform some computation and after finishing the computation they all synchronize and exchange data. This program execution model is valid if parallelism is obtained through data decomposition. With this final assumption T_{par} can be expressed as [37]

† Here it is assumed that every processor in the system contains exactly one process

[11] $T_{par} = \frac{T_{seq}}{p} + T_{calc.np} + T_{comm}$,

with

T_{seq} ≡ T_{par}(p=1,n);
$T_{calc.np}$ ≡ $T_{calc.np}$(p,n; τ_{calc},topology);
T_{comm} ≡ T_{comm}(p,n;$\tau_{startup}$,τ_{comm},topology).

We have slightly adapted the formulations of Basu et al., by describing the effect of their system utilisation parameter U*p* by $T_{calc.np}$.

We define T_{seq}, the execution time of the sequential algorithm, to be the execution time of the parallel algorithm on 1 processor, and not as the execution time of the fastest known sequential implementation. $T_{calc.np}$ describes all calculations that cannot be performed completely in parallel ('np' stands for non-parallel). Effects of load imbalance will, among others, appear in this term. T_{comm} is the total communication time of all cycles of the parallel program.

In the next sections we discuss a number of possible data decompositions and network topologies. The expressions for T_{par} will be written in increasing detail as a function of the variables and system parameters. Finally, a measurement of the system parameters will give numerical values of T_{par} as a function of p and n. On basis of this information a specific parallelization strategy (i.e. a data decomposition and network topology combination) is chosen.

3.2 Decomposition

One iteration of the PCGNR algorithm‡ contains 3 vector updates ($\mathbf{x}_{k+1}$, $\mathbf{r}_{k+1}$, $\mathbf{p}_{k+1}$), 3 vector inner products ($[(\mathbf{A}^H\mathbf{r}_k)^H\ \mathbf{s}_k]$, $[(\mathbf{A}\mathbf{p}_k)^H\ (\mathbf{A}\mathbf{p}_k)]$, $[\mathbf{r}_k\ \mathbf{r}_k]$), and 2 + 2m matrix vector products ($\mathbf{A}^H\mathbf{r}_k$, $\mathbf{A}\mathbf{p}_k$, and 2m for the polynomial preconditioning). Table 1 gives the execution times on a single processor for these operations. The total execution time for one iteration is

[12] $$T_{seq}(n) = \left(16(m+1)n^2 + (44-4m)n - 6\right)\tau_{calc}\ ,$$

where the squareroot operation (norm of $\mathbf{r}_k$) and two divisions (α_k, β_k) are neglected.

The PCGNR method possesses three types of data: scalars, vectors and matrices. All operations are performed on vectors and scalars. The matrices **A** and **M** remain unchanged, and appear only in matrix vector products. Therefore it is possible to define a static decomposition of the matrices. This prevents that large pieces of the matrices have to be sent over the network. In the sequel three different decompositions of the matrix will be discussed: the row-block decomposition, the columnblock decomposition, and the grid decomposition.

‡ From here on we will calculate the time complexity of one iteration step of the PCGNR algorithm. The initialization time is comparable to the time needed for one iteration (due to the same number of matrix vector products). Therefore it is not necessary to include it in the analysis. Furthermore, if the number of iterations becomes large, the initialization time can be neglected compared to the total iteration time.

Since we want to perform as many calculations as possible in parallel, the vectors are also decomposed. For parallel vector operations just one decomposition is possible. The vector is divided in equal parts, which are assigned to the processors in the network. As will become clear in the next paragraphs, this decomposition does not always match the matrix decomposition. Therefore, during the execution of the algorithm, the decompostion of the vectors changes. The vector decomposition is dynamic, parts of the vector will be sent over the network during the execution of the parallel PCGNR.

Operation	T_{seq}
vector update (vu)	$T_{seq}^{vu}(n) = 8n\,\tau_{calc}$
vector inner product (vi)	$T_{seq}^{vi}(n) = (8n-2)\,\tau_{calc}$
matrix vector product (mv)	$T_{seq}^{mv}(n) = (8n^2-2n)\,\tau_{calc}$

Table 1: The execution times of the three basic operations. The matrix is of size n×n and the vectors are of length n; all numbers are complex.

The rest of this section presents the parallel vector operations and their time complexity, followed by the parallel matrix vector product for the three different matrix decompositions. Appendix A introduces our symbolic notation to visualize the decompositions.

The vector operations

The parallel vector update is performed as depicted in diagram 1. The vector update is performed completely in parallel, provided that the scalar "factor" (α_k or β_k) is known in every processor. The result of the vector update is evenly distributed over the processors. This vector is either used as input for a matrix vector product ($\mathbf{r}_{k+1}$ and $\mathbf{p}_{k+1}$) or is further processed after the PCGNR algorithm has terminated ($\mathbf{x}_{k+1}$). The computing time for this parallel vector update (complex numbers) is

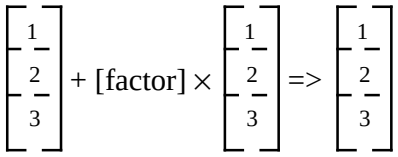


Diagram 1: parallel vector update

$$[13] \qquad T_{par}^{vu}(p,n) = 8\left\lceil \frac{n}{p} \right\rceil \tau_{calc} = \frac{T_{seq}^{vu}(n)}{p} + 8\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p} \right)\tau_{calc} \ ,$$

with $\lceil x \rceil$ the ceiling function of x. Equation [13] is of the form of equation 11. The parallel vector update, implemented as in diagram 1, does not have to communicate data; $T_{comm} = 0$. The non-parallel part is a pure load balancing effect:

$$[14] \qquad T_{np.calc}^{vu}(p,n) = 8\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p} \right)\tau_{calc} \ .$$

If the length of the vector n is not precisely divisible by the number of processors p, than some processors receive pieces of the vector with size $\lceil n/p \rceil$, and some with size $\lceil n/p \rceil$-1. This introduces a load imbalance in the system, of which the effect is described by equation 14. Note that for a large grain size n/p $T_{np.calc}$ is very small compared to T_{seq}/p.

The parallel inner product is shown in diagram 2:

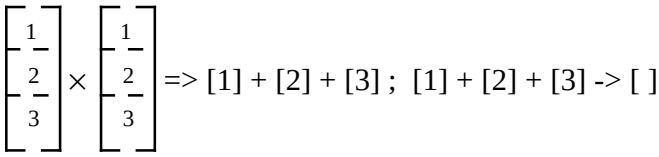


Diagram 2: the parallel innerproduct.

Every processor calculates a partial innerproduct, resulting in a partial sum decomposition of scalars. These scalars are accumulated in every processor and summed, resulting in an inner product, known in every processor. This is necessary because the results of parallel inner products, the numbers α_k and β_k, are needed in the parallel vector update as the scalar "factor" The total time for the parallel innerproduct is:

[15]
$$T^{vi}_{par}(p,n) = \left(8\left\lceil \frac{n}{p} \right\rceil - 2\right)\tau_{calc} + t_{ca.calc} + t_{ca}$$
$$= \frac{T^{vi}_{seq}(n)}{p} + \left(t_{ca.calc} - 2\left(1 - \frac{1}{p}\right)\tau_{calc}\right) + 8\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p}\right)\tau_{calc} + t_{ca}$$

where t_{ca} is the time for a complex accumulate, the total communication time to send the complex partial inner products resident on each processor to all other processors. The $t_{ca.np}$ is a computing time introduced by summing the partial inner products after (or during) the scalar accumulation. For this parallel vector inner product we find

[16]
$$T^{vi}_{comm}(p,n) = t_{ca} \ ,$$

and

[17]
$$T^{vi}_{calc.np}(p,n) = \left(t_{ca.calc} - 2\left(1 - \frac{1}{p}\right)\tau_{calc}\right) + 8\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p}\right)\tau_{calc} \ .$$

The non-parallel part consists of two terms. The second term in equation 17 is the load imbalance term. The first term describes the sommations in the parallel vector inner product that cannot be performed completely in parallel. The exact form of this term depends on the way in which the complex accumulate operation is implemented, and therefore depends on the topology of the network. This is also true for T_{comm}.

The matrix vector products

We will consider the matrix vector product for three different decompositions of the matrix; the row-block -, column-block -, and grid decomposition. These matrix decompositions dictate how the argument - and result vector of the matrix vector product must be decomposed. This vector decomposition will usually not match the decomposition as considered above. Therefore, the parallel matrix vector product requires pre- and post processing of the vectors, as will become clear in the next paragraphs.

- The rowblock decomposition.

The row-block decomposition is achieved by dividing **A** in blocks of rows, with every block containing $\lceil N/p \rceil$ or $(\lceil N/p \rceil - 1)$ consecutive rows, and assigning one block to every processing element. This is drawn schematically in diagram 3. Note that **A** is symmetric so that $\mathbf{A}^H$ is also decomposed in row-block.** The kernel of the parallel matrix vector product (**A** × vector *and* $\mathbf{A}^H$ × vector) is shown in diagram 4.

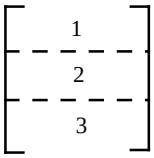


Diagram 3: the rowblock decomposition

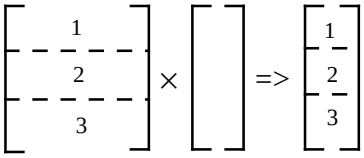


Diagram 4: the kernel of the parallel matrix vector product, with rowblock decomposition of the matrix.

The argument vector must reside in memory of every processing element. However, this vector is always the result of a vector update, or a previous matrix vector product, and is distributed over the processing elements (see diagram 1 and 4). Therefore, before calculating the matrix vector product, every processing element must gather the argument vector. The result is already decomposed in the correct way for further calculations (inner products, vector updates, or matrix vector products). The diagram for the total parallel matrix vector product, with vector preprocessing is drawn in diagram 5.

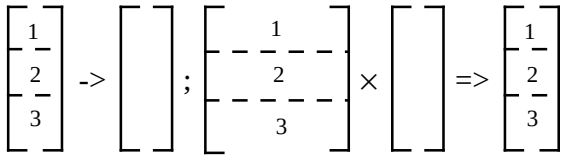


Diagram 5: the total parallel matrix vector for a rowblock decomposed matrix.

The vector is gathered firstly; all processors send their own part of the vector to all other processors. Next the matrix vector product is performed. The total execution time for this operation is (*mv;rb* means matrix vector product in row-block decomposition)

$$\text{[18]} \qquad \begin{aligned} T_{par}^{mv;rb}(p,n) &= \left\lceil \frac{n}{p} \right\rceil (8n-2)\tau_{calc} + t_{vg} \\ &= \frac{T_{seq}^{mv}(n)}{p} + (8n-2)\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p} \right)\tau_{calc} + t_{vg} \end{aligned} ,$$

with t_{vg} the time needed for the vector gather operation. The non-parallel time T_{np} only consists of a load balancing term, and the only contribution to T_{comm} is the vector gather operation.

- The columnblock decomposition

** If **A** is not symmetric, $\mathbf{A}^H$ would be decomposed in columnblock (see next paragraph) and the analysis changes slightly, also incorporating elements from columnblock decomposition. It turns out that in the non-symmetric case, the time complexity of the rowblock and columnblock decomposition is equal (data not shown).

Column-block decomposition is achieved by grouping ⌈N/p⌉ or (⌈N/p⌉-1) consecutive columns of **A** into blocks and distributing these blocks among the processing elements (see diagram 6).

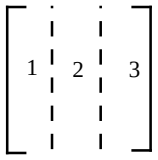


Diagram 6: the column-block decomposition.

For a symmetric matrix the Hermitian of the matrix is also decomposed in column-block. The computational kernel for the column-block decomposition is very different from the row-block kernel, and is drawn in diagram 7. The result of the matrix vector product is a partial sum decomposition that must be accumulated and scattered such that the resulting vector is properly distributed for further calculations. This vector is either an input for another matrix vector product, or for a vector operation. All require the vector to be distributed evenly over the processors (see diagram 1, 2, and 7). This defines the post processing communication step, which we call a partial vector accumulate, and some floating-point operations in evaluating the partial sums. The total parallel matrix vector product is drawn in diagram 8.

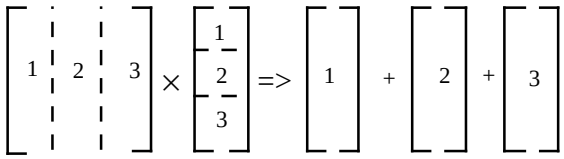


Diagram 7: the computational kernel of the parallel matrix vector product for a rowblock decomposed matrix.

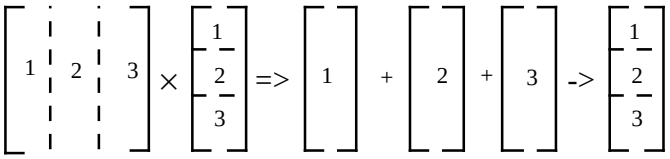


Diagram 8: the total parallel matrix vector product for a rowblock decomposed matrix.

The total time for this operation is

[19]
$$\begin{aligned} T_{par}^{mv;cb}(p,n) &= \left(8n\left\lceil\frac{n}{p}\right\rceil - 2n\right)\tau_{calc} + t_{va.calc} + t_{va} \\ &= \frac{T_{seq}^{mv}(n)}{p} + \left[t_{va.calc} - 2n\left(1-\frac{1}{p}\right)\tau_{calc}\right] + 8n\left(\left\lceil\frac{n}{p}\right\rceil - \frac{n}{p}\right)\tau_{calc} + t_{va} \end{aligned},$$

where t_{va} is the time to accumulate and scatter the resulting vector, and $t_{va.calc}$ the time to evaluate the partial sums. Here T_{np} consists of a load balancing term (the 3rd term) and a term for the summations in the partial vector gather operation (the 2nd term). T_{comm} is the communication time for the partial vector accumulate operation.

- The grid decomposition

The grid decomposition is a combination of rowblock and columnblock decomposition. The (square) matrix **A** is decomposed in p square blocks, which are distributed among the processing elements. The grid decomposition is drawn in diagram 9.

The Hermitian of **A** also is grid decomposed, therefore the constrained of a symmetric matrix is not needed for this decomposition. The computational kernel for the parallel matrix vector product is shown in diagram 10 (the kernel for the parallel Hermitian matrix vector product looks the same). The argument vector is decomposed as a mix form between the decomposition of the argument vector in the rowblock and columnblock matrix decomposition. The vector is divided in $p^{1/2}$ equal parts, and these parts are assigned to $p^{1/2}$ processing elements. The vector is scattered among the processing elements containing a row of blocks of the matrix. The result vector is scattered among the processing elements in the block column direction, partially sum decomposed among the processing elements in block row direction. Both the argument and result vector are not properly decomposed for the vector updates and inner products, nor for another matrix vector product. Therefore the total matrix vector product contains the kernel and two communication routines, see diagram 11.

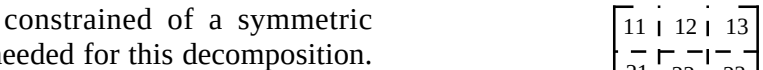


Diagram 9: the grid decomposition

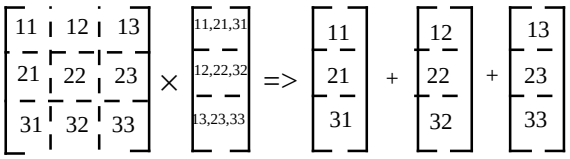


Diagram 10: the computational kernel for the parallel matrix vector product for a grid decomposed matrix.

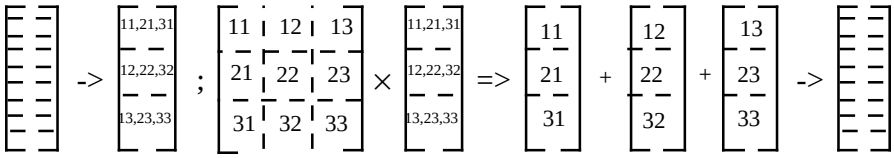


Diagram 11: the total parallel matrix vector product for a grid decomposed matrix.

The first communication routine is a partial vector gather operation, the second is a partial vector accumulate, followed by a scatter operation. The total time for the parallel matrix vector product can now be expressed as follows:

$$
\begin{aligned}
T_{par}^{mv;g}(p,n) &= \left\lceil \frac{n}{\sqrt{p}} \right\rceil \left(8 \left\lceil \frac{n}{\sqrt{p}} \right\rceil - 2 \right) \tau_{calc} + t_{pva.calc} + t_{pvg} + t_{pva} \\
&= \frac{T_{seq}^{mv}(n)}{p} + \left[t_{pva.calc} - 2n \left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil \frac{1}{n} - \frac{1}{p} \right) \tau_{calc} \right] , \\
&\quad + 8 \left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil^2 - \frac{n^2}{p} \right) \tau_{calc} + t_{pvg} + t_{pva}
\end{aligned}
\qquad [20]
$$

t_{pvg} is the time for the partial vector gather operation, t_{pva} the time for the partial vector accumulate, and $t_{pva.calc}$ the time to evaluate the partial sums after or during the vector accumulation.

Once more T_{np} consists of a loadbalancing term and floating-point operations that are not performed completely in parallel, and T_{comm} is the time needed in the communication of the pre- and post processing steps.

A comparison

At this point the execution time of the parallel PCGNR for the three matrix decompositions can be compared. The only difference, at this level of abstraction, appears in the execution time of the matrix vector product. Table 2 summarizes T_{comm} and T_{np} of the matrix vector product for the three decompositions, where T_{np} is further subdivided in the load balance term and the other non-parallel floating-point operations.

decompostion	T_{np} load balance	T_{np} other non-parallel floating- point operations	T_{comm}
rowblock	$(8n-2)\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p}\right)\tau_{calc}$	none	t_{va}
columnblock	$8n\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p}\right)\tau_{calc}$	$t_{va.calc} - 2n\left(1-\frac{1}{p}\right)\tau_{calc}$	t_{va}
grid	$8\left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil^2 - \frac{n^2}{p}\right)\tau_{calc}$	$t_{pva.calc} - 2n\left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil \frac{1}{n} - \frac{1}{p}\right)\tau_{calc}$	$t_{pvq} + t_{pva}$

Table 2: T_{np} and T_{comm} of the matrix vector product, for three different matrix decompositions, T_{np} is further divided in a load balancing term and other non-parallel computations

In practical situations it can be assumed that a load imbalance is present. In that case

[21] $T_{np}^{mv;rb} < T_{np}^{mv;cb}$ (for perfect load balance $T_{np}^{mv;rb} \leq T_{np}^{mv;cb}$)

therefore, if

[22] $T_{comm}^{mv;rb} \leq T_{comm}^{mv;cb}$, (for perfect load balance $T_{comm}^{mv;rb} < T_{comm}^{mv;cb}$)

it follows that

[23] $T_{par}^{mv;rb} < T_{par}^{mv;cb}$.

This means that if equation 22 holds, the implementation of the PCGNR algorithm with a rowblock decompostion of the matrix is always faster then an implementation with a columnblock decompostion. This is true for any parallel computing system as defined in section 3.1.

During a vector gather operation, appearing in the rowblock decomposition (see diagram 5), every processor must send one packet of maximum $\lceil n/p \rceil$ complex numbers to all other processors. In the vector accumulate operation of the columnblock decompostion (see diagram 8) every processor must send a different packet of maximum $\lceil n/p \rceil$ complex numbers to every other processor.

On a fully connected network, where all processors are connected to each other and the links operate at the same speed, the vector gather and vector accumulate operation both take the same time, i.e. the time to send one packet of $\lceil n/p \rceil$ complex numbers over a link. In this case $t_{vg} = t_{va}$ and equation 22 holds.

For less rich network topologies, where processor pairs are connected via one or more

other processors, the vector accumulate takes longer time. This can be seen as follows. During the vector gather operation, after receiving and storing a packet, a processor can pass this packet to another processor, which also stores it and passes it along. The vector gather operation now takes say s times the time to send one packet of $\lceil n/p \rceil$ complex numbers over a link, where s depends on the specific network topology.

In the vector accumulate operation all processors must send different messages to all other processors. As with the vector gather operation, this can be achieved in s steps, but now the packet size is larger. A processor, say j, receives at some time during the accumulate operation a packet originally sent by processor i. This packet contains a part of the complete vector (size $\lceil n/p \rceil$ complex numbers) destined for processor j, and the packets for the processors which receive data from processor i via processor j. The exact size of all the messages depends strongly on the network topology and implementation of the vector accumulate, but the mean packet size in the s steps must be larger then $\lceil n/p \rceil$ complex numbers. The vector accumulate can also be achieved with packages of $\lceil n/p \rceil$ complex numbers, but than the number of steps needed to complete the operation must be larger then s.

Therefore, in case of load imbalance, equation 23 always holds, and without load imbalance equation 23 holds if the network is not fully connected (which is almost always the case). Without specific knowledge of the computing system we can rule out PCGNR for symmetric matrices with columnblock decomposition of the matrix.

3.3 Topology

As a consequence of the data decomposition (parts of) the vectors and scalars must be communicated between processors. In this section we will derive expressions for the execution times of these communication routines for several network topologies.

Table 3 lists the topologies that can be realised with transputers, assuming that every processing element consists of 1 transputer and that the network contains at least 1 'dangling' (free) link to connect the network to a host computer.

The topologies are categorized in four groups; tree's, (hyper)cubes, meshes and cylinders. Two tree topologies can be build, the binary and ternary tree. The number of hypercubes is also limited. Only the cube of order 3 is in this group, cubes of lower order are meshes, whereas cubes of higher order cannot be realised with single transputer nodes. Transputers are usually connected in mesh and cylinder topologies. Note that the pipeline and ring topology are obtained by setting $p_2 = 1$. The torus topology (mesh with wrap around in both directions) is not possible because this topology, assembled from transputers, contains no dangling links.

We will examine the time complexity of the communication routines mentioned previously, plus overhead, for two topologies; a bidirectional ring and a square ($p_1=p_2$) cylinder with bidirectional links. The rowblock decomposition is analysed on the ring, the grid decomposition is considered in conjunction with the $p_1=p_2$ cylinder (from now on referred to as cylinder). We have also analysed the rowblock decomposition on the cylinder and the binary tree. The total execution time on the cylinder is higher than on the ring and the binary tree. The time on the last two is comparable (data not shown).

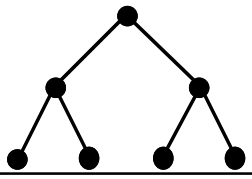
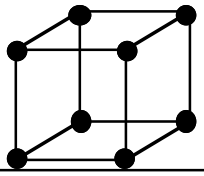
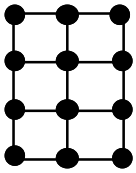
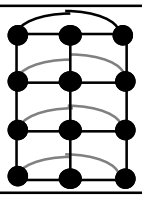
Topology	Order	Number of processors (p)	Comment	Example
k-tree	d	$\frac{1}{k-1}k^{d+1} - \frac{1}{k-1}$	$2 \le k \le 3$	binary tree, k = 2 
Hypercube	d	2^d	$d \le 3$	
Mesh		$p_1 x p_2$	p_1: number of processors in a row. p_2: number of processors in a column.	
Cylinder		$p_1 x p_2$	Wrap around in p_1 direction	

Table 3: Overview of topologies that can be realised with transputers. The black dots are processing elements (PE), every PE consists of 1 transputer. The network must contain at least 1 free link for connection to a host computer. All links are bidirectional.

The rowblock decomposition on a bidirectional ring

The parallel PCGNR with rowblock decomposition of the matrix contains a vector gather operation and a complex accumulate operation. During the vector gather operation (see diagram 5) every processor receives from any other processor in the network a part of the vector, which is subsequently stored. After the vector gather operation each processor has a copy of the total vector in its local memory. On the bidirectional ring this is achieved as follows:
1) In the first step every processor sends its local part of the vector to the left and the right processor and, at the same time, receives from the left and the right processor their local part.
2) In the following steps, the parts received in a previous step, are passed on from left to right and vice versa, and in parallel, parts coming from left and right are received and stored.
After $\lfloor p/2 \rfloor$ steps (with $\lfloor x \rfloor$ the floor function of x), every processor in the ring has received the total vector. The execution time for the vector gather operation on a bidirectional ring is (assuming double precision numbers, i.e 16 bytes for a complex number)

$$[24] \quad t_{vg}^{ring} = \left\lfloor \frac{p}{2} \right\rfloor \left(\tau_{startup} + 16 \left\lceil \frac{n}{p} \right\rceil \tau_{comm} \right).$$

The time for the complex accumulate operation (see diagram 2) is easily derived from the previous analysis. Instead of a complex vector of $\lceil n/p \rceil$ elements a complex number is sent. After every communication step, the newly received partial inner products are added to the partial inner product of the receiving processor. In total p complex numbers are added. This

leads to the following expressions for t_{ca} and $t_{ca.calc}$:

[25] $$t_{ca}^{ring} = \left\lfloor \frac{p}{2} \right\rfloor \left(\tau_{startup} + 16\tau_{comm} \right);$$

[26] $$t_{ca.calc}^{ring} = 2(p-1)\tau_{calc} \ .$$

Grid decomposition on a cylinder

The parallel PCGNR with grid decomposition of the matrix contains a partial vector gather operation, a partial vector accumulate operation, and a complex accumulate operation. Derivation of the execution time of these operations on a cylinder topology is not as straightforward as for the previous case. First we must decide in which direction the wrap around in the cylinder is made; horizontally or vertically. We assume a one to one relationship between the grid decomposition of the matrix (see diagram 9) and the distribution of the submatrices over the processor network, e.g. block 23 of the decomposed matrix resides in memory of processor 23 (row 2, column 3) of the cylinder. In this case the partial vector gather operation (see diagram 11) results in a vector, scattered over every row of the cylinder (this is also true for the Hermitian matrix vector product if the the symmetry property of **A** is applied). This suggests a 'column wise' scattering of the original vector over the grid, see diagram 12.

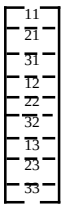


Diagram 12: the column wise decomposition of a vector

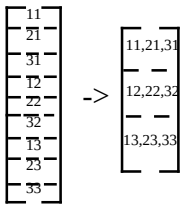


Diagram 13: The partial vector gather, with a column wise decomposed vector.

The partial vector gather operation is once again drawn in diagram 13 which, implemented in this way, consists of $p^{1/2}$ simultaneous vector gathers along the columns of the cylinder. Every processor in the column must receive a package of $\lceil n/p \rceil$ complex words of every other processor in the column.

The vector accumulate can be separated into two communication steps, see diagram 14. The first step is a communication in the horizontal direction. The vector, which is sum decomposed in the horizontal direction is accumulated in the diagonal processors of the cylinder (performing the summations on the diagonal processors[§§]). The vector is now decomposed over the diagonal of the cylinders in parts of $\lceil n/(p^{1/2}) \rceil$ elements. These parts are scattered in the second step in vertical direction, resulting in the 'column wise' scattering of the original vector. The first step implies that every processor on the diagonal must receive packages of $\lceil n/(p^{1/2}) \rceil$ complex numbers from every

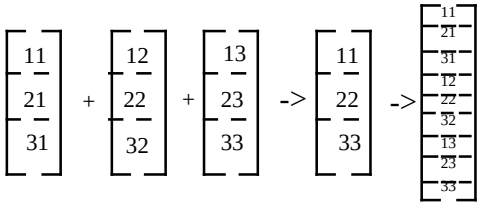


Diagram 14: The vector accumulate, separated into two different communication routines.

§§ The summations can be postponed until all data is present on all processors. Then all summations can be performed completely in parallel. Unfortunately this leads to much more communication, thus increasing the total execution time of the partial vector accumulate.

processor in the row of the grid. In the second step, the diagonal processors must sent packages of $\lceil n/p \rceil$ complex words to all processors in their column.

Note that both in the partial vector gather and in the vector accumulate the grid can be viewed as $p^{1/2}$ independently operating pipelines or rings (depending on the wrap around) containing $p^{1/2}$ processors. In the first step of the vector accumulate (see diagram 14) the largest packages must be sent (in row wise direction). Therefore it is most advantageous to make a wrap around in horizontal direction, creating rings in row wise direction.

Now we can derive expression for t_{pvg} and t_{pva}. The partial vector gather is a gather operation of packages of $\lceil n/p \rceil$ complex words on $p^{1/2}$ simultaneously operating bidirectional pipelines each containing $p^{1/2}$ processors. Basically this is implemented in the same way as the vector gather operation on the ring (see previous section), but now a total of $p^{1/2}$-1 steps are required to get the messages from every processor in the pipeline to every other processor. Therefore the execution time of the partial vector gather operation on the cylinder is

[27] $$t_{pvg}^{cylinder} = (\sqrt{p}-1)\left(\tau_{startup} + 16\left\lceil \frac{n}{p} \right\rceil \tau_{comm} \right) .$$

The vector accumulate operation is an accumulation to the processors on the diagonal of the cylinder of a sum decomposed vector of $\lceil n/(p^{1/2}) \rceil$ complex words on $p^{1/2}$ simultaneously operating bidirectional rings of $p^{1/2}$ processors, followed by scattering of packages of $\lceil n/p \rceil$ complex words from a processor to all other processors in $p^{1/2}$ simultaneously operating pipelines containing $p^{1/2}$ processors, therefore

[28] $$t_{pva}^{cylinder} = \left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \left(\tau_{startup} + 16\left\lceil \frac{n}{\sqrt{p}} \right\rceil \tau_{comm} \right) + (\sqrt{p}-1)\left(\tau_{startup} + 16\left\lceil \frac{n}{p} \right\rceil \tau_{comm} \right).$$

The first step of the partial vector accumulate operation introduces the calculations that should be accounted for by $t_{pva.calc}$. After every communication step the received vectors of $\lceil n/p^{1/2} \rceil$ complex words are added to the partially accumulated vector in the diagonal processor. A total of $p^{1/2}$ vectors must be added, therefore

[29] $$t_{pva.calc}^{cylinder} = 2(\sqrt{p}-1)\left\lceil \frac{n}{\sqrt{p}} \right\rceil \tau_{calc} .$$

The complex accumulate is straightforward, the partial inner products are first accumulated in the horizontal direction, followed by an accumulation in the vertical direction. In both sweeps $p^{1/2}$ complex number are added. This leads to the following expressions for t_{ca} and $t_{ca.calc}$:

[30] $$t_{ca}^{cylinder} = \left\lfloor \frac{\sqrt{p}}{2} \right\rfloor \left(\tau_{startup} + 16\,\tau_{comm} \right) + (\sqrt{p}-1)\left(\tau_{startup} + 16\,\tau_{comm} \right) ;$$

[31] $$t_{ca.calc}^{cylinder} = 4(\sqrt{p}-1)\,\tau_{calc} .$$

A comparison

The derivation of the expression for the communication - and non-parallel routines, as a function of the systemparameters, allows us to compare T_{par} of the PCGNR method for two different parallelization strategies: the row-block decomposition on a ring and the grid decomposition on a cylinder.

Here we will compare T_{np} and T_{comm} of both parallelization strategies in some limited cases. In the next section numerical values for the system parameters are presented, and a detailed comparison will be made. From section 3.1 and 3.2 it follows that

[32] $$T_{np}^{PCGNR} = 3T_{np}^{vu} + 3T_{np}^{vi} + (2m+2)T_{np}^{mv} \ ,$$

and

[33] $$T_{comm}^{PCGNR} = 3T_{comm}^{vu} + 3T_{comm}^{vi} + (2m+2)T_{comm}^{mv} \ .$$

Combining the expressions of the previous section with equations [32, 33] results in:

[34] $$\left(T_{np}^{PCGNR}\right)_{ring}^{rowblock} = \left[(16(m+1)n + 44 - 4m)\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p}\right) + 6(p + \frac{1}{p} - 2)\right]\tau_{calc} \ ;$$

[35] $$\left(T_{comm}^{PCGNR}\right)_{ring}^{rowblock} = (2m+5)\left\lfloor \frac{p}{2} \right\rfloor \tau_{startup} + 16\left\lfloor \frac{p}{2} \right\rfloor \left((2m+2)\left\lceil \frac{n}{p} \right\rceil + 3\right)\tau_{comm} \ ;$$

[36] $$\left(T_{np}^{PCGNR}\right)_{cylinder}^{grid} = \left[16(m+1)\left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil^2 - \left(\frac{n}{\sqrt{p}}\right)^2\right) + 6\left(\left\lceil \frac{n}{p} \right\rceil - \frac{n}{p}\right) + (2m+2)\left((2\sqrt{p} - 4)\left\lceil \frac{n}{\sqrt{p}} \right\rceil + 2\frac{n}{p}\right) + 6(2\sqrt{p} + \frac{1}{p} - 3)\right]\tau_{calc} \ ;$$

[37] $$\left(T_{comm}^{PCGNR}\right)_{comm}^{cylinder} = \left[(2m+5)\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + (4m+7)(\sqrt{p} - 1)\right]\tau_{startup} + 16\left[(2m+2)\left(\left\lceil \frac{n}{\sqrt{p}} \right\rceil \left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + 2(\sqrt{p} - 1)\left\lceil \frac{n}{p} \right\rceil\right) + 3\left(\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + \sqrt{p} - 1\right)\right]\tau_{comm} \ .$$

In case of a perfect load balance it follows from equation 34 and 36 that (assuming n >> p)

[38] $$\left(T_{np}^{PCGNR}\right)_{ring}^{rowblock} = O(p)\tau_{calc} \ ,$$

and

[39] $$\left(T_{np}^{PCGNR}\right)_{cylinder}^{grid} = O(n)\tau_{calc} \ .$$

With a load imbalance in the system, T_{np} for the grid-cylinder combination has the same order of magnitude as with a perfect load balance (equation 39). However, a load imbalance has a strong impact on the row-block-ring combination; T_{np} now becomes

$$[40] \qquad \left(T_{np}^{PCGNR}\right)_{ring}^{rowblock} = O(n)\tau_{calc} \ ,$$

which is the same order of magnitude as the grid-cylinder combination.The order of magnitude of T_{comm} follows from equation 35 and 37, and is

$$[41] \qquad \left(T_{comm}^{PCGNR}\right)_{ring}^{rowblock} = O(p)\tau_{startup} + O(n)\tau_{comm} \ ,$$

and

$$[42] \qquad \left(T_{comm}^{PCGNR}\right)_{cylinder}^{grid} = O(\sqrt{p})\tau_{startup} + O(n)\tau_{comm} \ .$$

Equations 39 to 42 show that T_{np} and T_{comm} for the rowblock-ring combination and the grid-cylinder combination are comparable. Numerical values of the system parameters are needed to decide which combination is better, and how much better. Furthermore, we must keep in mind that T_{par} for the PCGNR implementation is mainly determined by T_{seq}/p, with an order of magnitude

$$[43] \qquad \frac{T_{seq}}{p} = O\left(\frac{n^2}{p}\right)\tau_{calc} \ .$$

Using the orders of magnitude as derived above, we can find a crude expression for the efficiency ε of the parallel PCGNR, where we have taken the worst case T_{comm} of equation 41:

$$[44] \qquad \varepsilon = \frac{T_{seq}}{pT_{par}} \approx \frac{1}{1 + O\left(\frac{p}{n}\right)\left(1 + \frac{\tau_{comm}}{\tau_{calc}}\right) + O\left(\frac{p^2}{n^2}\right)\frac{\tau_{startup}}{\tau_{calc}}} \ .$$

Equation 44 shows that deviations from $\varepsilon = 1$ occur due to three sources. Let us first concentrate on the term involving τ_{comm}. This term can be identified with the communication overhead f_c defined by Fox et al.[35, section 3-5]. For a problem dimension d_p and a dimension of the complex computer d_c Fox et al. show that in general (in our notation)

$$[45] \qquad f_c = \begin{cases} \dfrac{constant}{\left(n^2/p\right)^{1/d_p}} \dfrac{\tau_{comm}}{\tau_{calc}} & \text{if} \quad d_c \geq d_p \\ p^{(1/d_c - 1/d_p)} \dfrac{constant}{\left(n^2/p\right)^{1/d_p}} \dfrac{\tau_{comm}}{\tau_{calc}} & \text{if} \quad d_c < d_p \end{cases} \ .$$

Thus, for the row-block-ring combination ($d_p = 2$ and $d_c = 1$) f_c is

[46] $f_c = constant \frac{p}{n} \frac{\tau_{comm}}{\tau_{calc}}$,

which is in agreement with equation 44. However, for the grid-cylinder combination ($d_c = 2$) one would expect

[47] $f_c = constant \frac{\sqrt{p}}{n} \frac{\tau_{comm}}{\tau_{calc}}$,

which disagrees with our result. The reason for this is the implementation of the partial vector accumulate. The first step of this operation (see diagram 14) cannot exploit the full dimensionality of the network, giving rise to an order of magnitude of T_{comm} as in equation 42.

The second term of the denominator in equation [44] is due to non-parallel computation (the grid cylinder combination) or due to load imbalance (the rowblock-ring combination) and has the same order of magnitude as the communication overhead. The last term in the denominator describes the efficiency reduction due to communication startup times. This term however is an order of magnitude smaller than the previous two.

As long as τ_{calc}, τ_{comm}, and $\tau_{startup}$ have the same order of magnitude, and n >> p, the efficiency of the parallel PCGNR algorithm can be very close to unity. In the next paragraph we will investigate this in more detail.

3.4 The hardware parameters

Our parallel computing system is a Meiko computing surface, consisting of 64 T800 transputers, each with 4 Mb RAM, hosted by a Sun sparc workstation. The configuration is described in more detail by Hoffmann and Potma [38]. The programming language is Occam 2 and the programming environment is the Occam Programming System (OPS). As described in section 3.1, the abstract model of the computing system consists of a specification of the network topology, and the three hardware parameters.

The OPS allows configuration of any possible network by specifying which transputer links of the available processors must be connected. The Meiko system services use this information to program the configuration chips in the computing surface. Furthermore, the programmer can specify exactly which tranputers in the computing surface must be booted. This allows to configure bidirectional rings of any number of transputers (up to 63, with 1 running the OPS) and cylinders with p = 4, 9, 16, 25, 36, and 49. The communication times were measured by sending different sized packets over the bidirectional links and measuring the total sending time. Fitting of the measurements to a straight line resulted in $\tau_{startup}$ = 13.3 μs, and τ_{comm} = 0.99 μs/byte.

The parameter τ_{calc} should not just incorporate the raw computing power of the floating point unit of the transputer, but also the overheads due to indexing and memory access. As a consequence it is virtually unachievable to define one unique value for τ_{calc}, because different operations, such as a single addition, or a vector update, give rise to different overheads. The abstraction of the computing system seems to coarse. Fortunately most floating-point operations in the PCGNR algorithm take place in the matrix vector products. Therefore we timed the complex matrix vector product, with all overheads, and used these results to derive τ_{calc}. The results is τ_{calc} = 1.62 μs/floating-point operation (double precision).

With the experimental values for the systemparameters a numerical calculation of T_{par} is

possible. To compare the rowblock-ring - and grid-cylinder combination a relative difference function rd(p,n) is defined:

$$[48] \qquad \mathrm{rd}(p,n) = \frac{(T_{par}^{PCGNR})_{ring}^{rowblock} - (T_{par}^{PCGNR})_{cylinder}^{grid}}{(T_{par}^{PCGNR})_{ring}^{rowblock} + (T_{par}^{PCGNR})_{cylinder}^{grid}} \; .$$

The sign of rb(p,n) tells which parallelization strategy is faster, the amplitude tells how big the relative difference between the two is.

Figures 2, 3, and 4 show rb for n = 1000, 10.000, and 100.000 and 1≤p≤100; figure 5 shows rb for n = 100.000 and 1≤p≤1000, where we considered the case m=0 (no preconditioning, m≠0 gives comparable results). In all cases rb strongly oscillates around zero, however the amplitude is very small. This means, from a time-complexity point of view, that both parallelization strategies are equivalent. Depending on the exact values of n and p, one is faster than the other. However, the relative difference in total execution time is very small. This is true as long as n>>p, which will be the normal situation.

Since the execution time T_{par} cannot justify a choice between the rowblock-ring - and the grid-cylinder combination, other criteria enter the discussion.

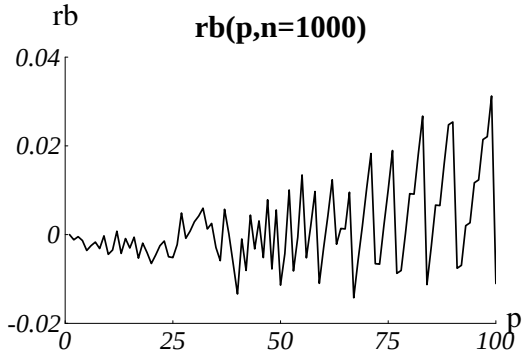


Figure 2: The relative difference function for n = 1000 and 1≤p≤100

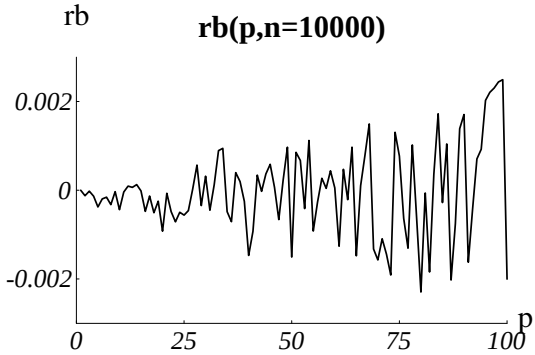


Figure 3: The relative difference function for n = 10000 and 1≤p≤100

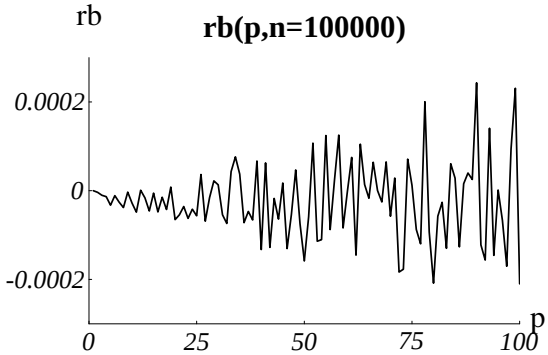


Figure 4: The relative difference function for n = 100000 and 1≤p≤100

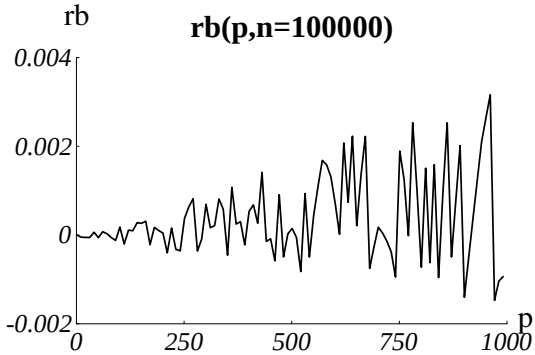


Figure 5: The relative difference function for n = 100000 and 1≤p≤1000

From an implementation point of view the rowblock-ring combination is preferable. The rowblock decomposition introduces just one communication routine: the gather operation. Parallel PCGNR with a grid decomposition on the other hand contains more, and more complex communication routines. Furthermore, from a user point of view the rowblock-ring combination has one important advantage. The ring can have any number of processors, and therefore the maximum number of free processors can always be used during production runs.

This is important in parallel computing environments where the users can request any number of processors. These considerations are all in favour of the rowblock-ring combination. Therefore it was decided to implement parallel PCGNR with a rowblock decomposition of the matrix, on a bi-directional ring of transputer.

Figure 6 show the theoretical efficiency of this parallel PCGNR. If the number of rows per processor is large enough the efficiency will be very close to unity. Therefore parallel PCGNR with rowblock decomposition, implemented on a bidirectional ring is very well suited for coarse grain distributed memory computers.

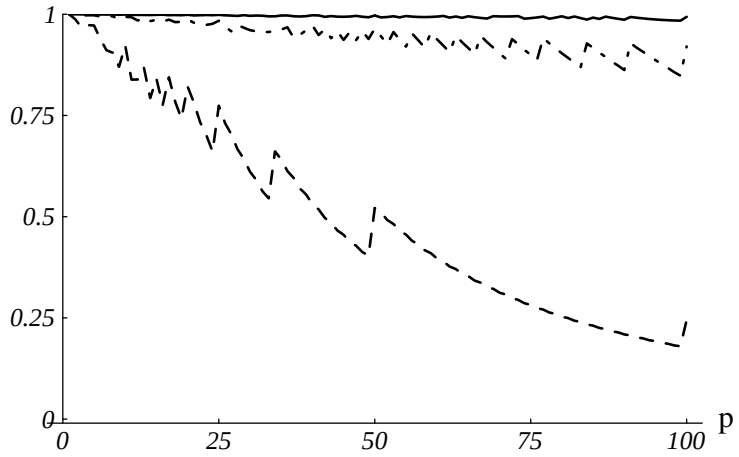


Figure 6: The theoretical efficiency for parallel PCGNR, m=0, with a rowblock decomposition of the matrix, implemented on a bidirectional ring of transputers. The dashed line is for n=100, the dot-dashed line is for n=1000 and the full line is for n=10000; p is the number of transputers

4] IMPLEMENTATION

The parallel PCGNR was implemented in Occam 2 [39] on the Meiko Computing Surface. The implementation consists of two parts: a harness containing the configuration information and communication libraries, and the actual PCGNR implementation.

Figure 7 shows the configuration of the transputers in the Computing Surface. The host transputer runs the Occam Programming System, and is connected with the host computer (a Sun Sparc II station). The root transputer is connected to the host transputer. The input/output from the parallel program to or from the screen, keyboard, or filing system of the host computer is performed by this root transputer. Furthermore, the root transputer, together with the p-1 slave transputers, run the harness and the parallel PCGNR. The host transputer is present for development, debugging and testing of the parallel program. During production runs the root transputer can be connected directly to the host computer.

The root and slave transputers all run the same two processes, a router and a calculator, see figure 8. Router processes on neighbouring transputers are connected via a channel. These channels are associated with hardware transputer links. The router process calls communication routines from a communication library. These routines, such as e.g. the vector gather operation, take data from local memory and send it to other routers, and process data that is received during the communication routine

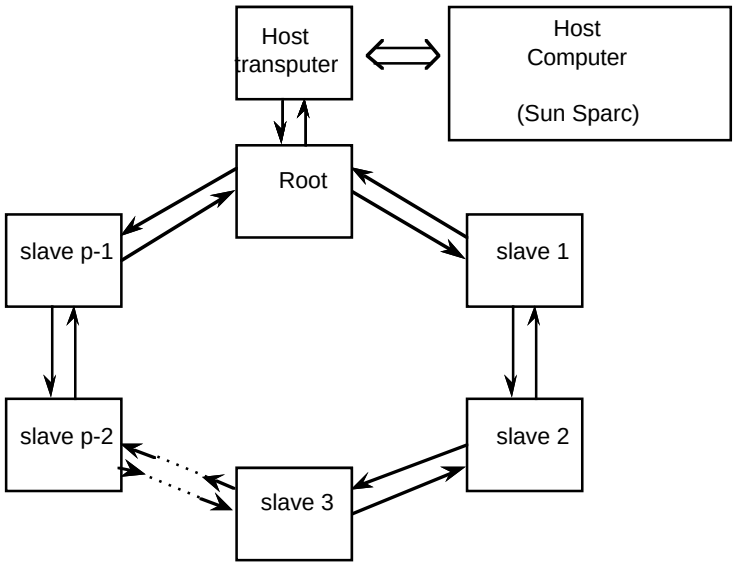


Figure 7: The configuration of the transputers

The calculator process performs the work on the decomposed data. This work is divided in cycles, as was described in section 3.1. At the end of every cycle a communication step occurs. The calculator sends a command, in the form of a single character, via the command channel to the router process. The router process receives this character, interprets it and issues the desired communication routine. During this communication step the calculator process is idle. After finishing the communication, the router process sends a 'ready' signal to the calculator process, which then proceeds with the next cycle in the algorithm. In principle the communication hardware and the CPU and FPU of the transputer can work in parallel, thus allowing to hide the communication behind calculations. We decided not to use this feature of the transputer, because total communication time is very small compared to calculation times.

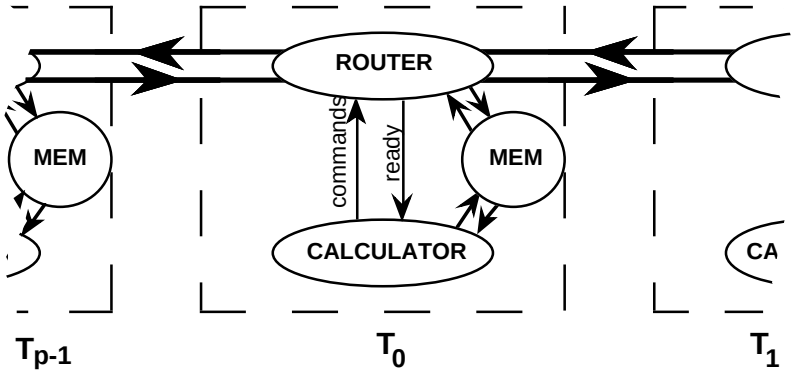


Figure 8: Main processes and streams in a transputer; MEM is the local memory, Router and Calculator are two OCCAM processes, and T_{p-1}, T_0, and T_1 are three neighbouring transputers in the bidirectional ring

This implementation keeps the matrix in memory. The maximum matrix that fits in the local memory of one transputer is n=495. On the full 63 transputer ring, the maximum matrix size is n=3885. For realistic, larger problems the matrix cannot be kept in memory. The matrix elements will then be calculated as soon as they are needed, using the definition in equation 5.

The main advantage of our implementation is the complete decoupling of the details of the communication and the "useful" work in the parallel program. The calculator code closely resembles the sequential code, with just some extra statements issuing commands to the router.

The algorithm is easily adapted by making changes in the calculator process. This is important for e.g. testing the influence of the parameter m in the preconditioner.

5] RESULTS

This section presents the results of two experiments. First we measured the performance of the implementation of the parallel PCGNR for m=0 (no preconditioning), by measuring the execution time of one iteration of the PCGNR algorithm, as a function of p and n. These results are compared with the theory of section 3. Secondly, we tested the convergence of the PCGNR algorithm, for typical matrices of the coupled dipole method, for m=0 and m=1.

5.1] Performance measurements

We have measured the execution time of one iteration of the parallel PCGNR, with m=0, for n = 60, 219, and 495 as a function of p, where $1 \leq p \leq 63$. Figures 9, 10, and 11 show the measured and calculated efficiencies of one iteration of the parallel PCGNR. Table 4 gives the error, in percents, between the theoretical and measured T_{par} as a function of p and n. Finally we measured T_{par} for the maximum problem on 63 transputers, i.e. n=3885. The error between theory and experiment was 3.4 %. The theoretical efficiency in this case is 0.98.

5.2] Convergence behaviour

The matrix **A**, as defined by equation 5 depends on the relative refractive index n_{rel} of the particle of interest, the wavelength of the incident light, and the position and size of the dipoles. To test the convergence behaviour of PCGNR for these matrices, the norm of the residual vector, as function of the iteration number k was measured, for some typical values of the parameters. Here we will show results for the currently largest possible matrix (n=3885, i.e. 1295 dipoles). The results for smaller matrices are comparable.

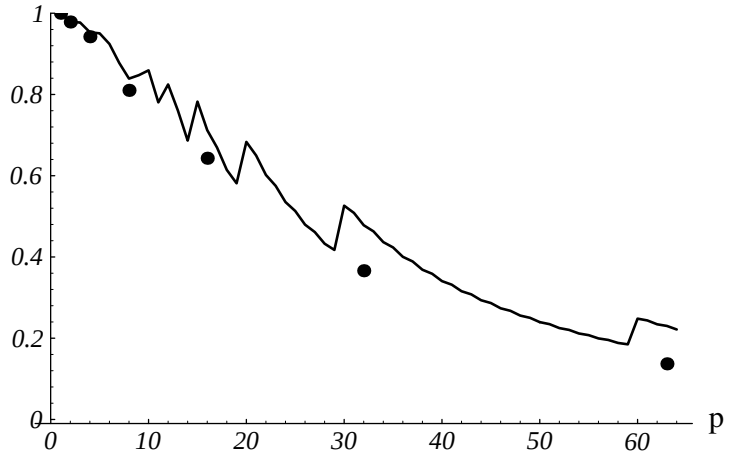


Figure 9: The efficiency on one iteration of the parallel PCGNR for n = 60, as a function of the number of processors. The black dots are the experimental results, the solid line is the theoretical efficiency.

The wavelenght of the incident light was set to λ = 488.0 nm (blue light), and the diameter of the dipoles to $\lambda/20$. The scattering particle was a sphere, the dipoles were put on a cubic grid with spacing $\lambda/20$. The relative refractive index was chosen to depict some representative material compounds of interest: n_{rel} = 1.05 and 1.5 to give the range of indices of

biological cells in suspension; n_{rel} = 1.33 + 0.05i, dirty ice; n_{rel} = 1.7 + 0.1i, silicates; and n_{rel} = 2.5 + 1.4i, graphite.

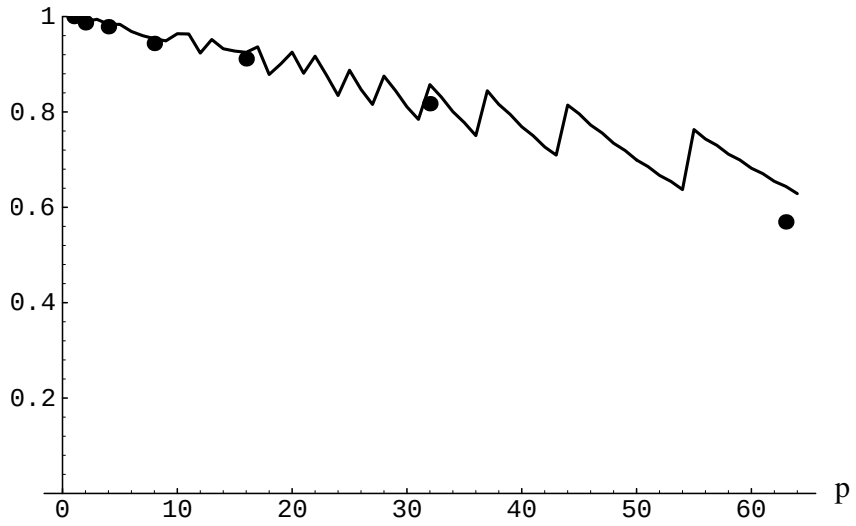


Figure 10: The efficiency on one iteration of the parallel PCGNR for n = 219, as a function of the number of processors. The black dots are the experimental results, the solid line is the theoretical efficiency.

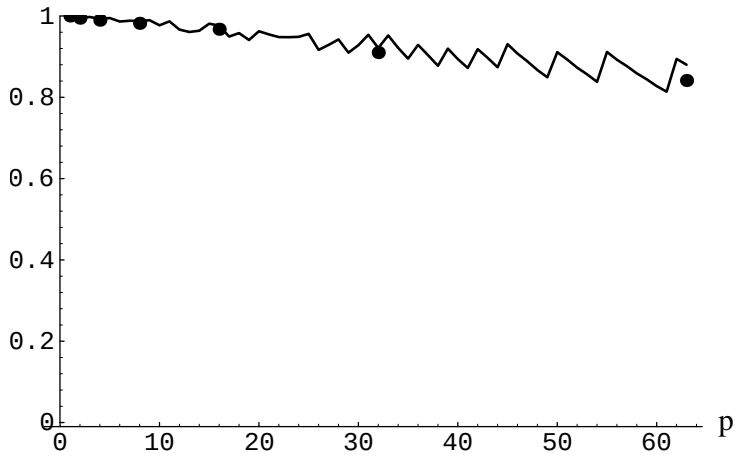


Figure 11: The efficiency on one iteration of the parallel PCGNR for n = 495, as a function of the number of processors. The black dots are the experimental results, the solid line is the theoretical efficiency.

		p						
		1	2	4	8	16	32	63
n	60	2.2	2.2	1.1	1.3	8.3	27.7	64.4
	219	0.5	0.8	1.0	1.5	2.0	5.4	13.6
	495	0.7	0.8	1.1	1.3	1.7	2.0	5.4

Table 4: The percentual error between the experimentally measured and theoretically calculated execution time of one iteration of the PCGNR, as a function of p and n.

Figures 12, 13, 14, 15, and 16 show the logarithm of the norm of the residual vector divided by the stopping criterion (ε times the norm of **b**) for n_{rel} = 1.05, n_{rel} = 1.5, dirty ice, silicates, and graphite respectively. If the measured function is smaller than zero, the iteration has converged. We tested for m=0 and for m=1 (first order von Neumann preconditioning).

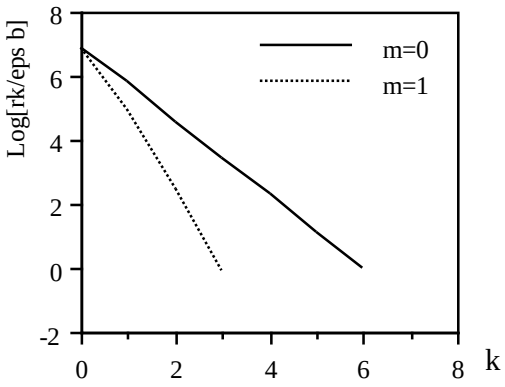


Figure 12: The norm of the logarithm of the residual vector $\mathbf{r}_k$ divided by the norm of **b** times ε, as a function of the iteration number, for n_{rel} = 1.05.

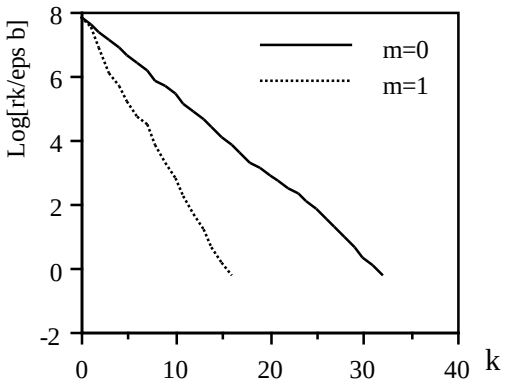


Figure 13: The norm of the logarithm of the residual vector $\mathbf{r}_k$ divided by the norm of **b** times ε, as a function of the iteration number, for n_{rel} = 1.5.

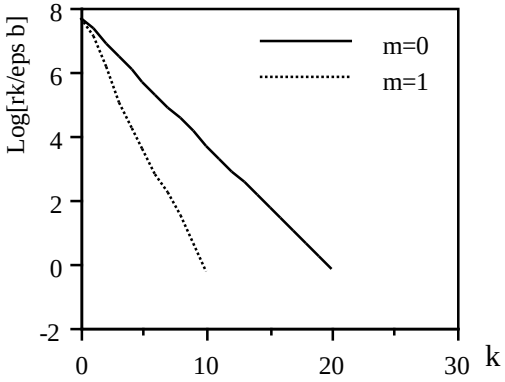


Figure 14: The norm of the logarithm of the residual vector $\mathbf{r}_k$ divided by the norm of **b** times ε, as a function of the iteration number, for n_{rel} = 1.33 + 0.05i (dirty ice).

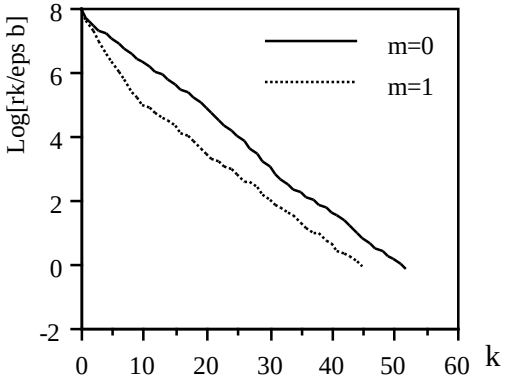


Figure 15: The norm of the logarithm of the residual vector $\mathbf{r}_k$ divided by the norm of **b** times ε, as a function of the iteration number, for n_{rel} = 1.7 + 0.1i (silicates).

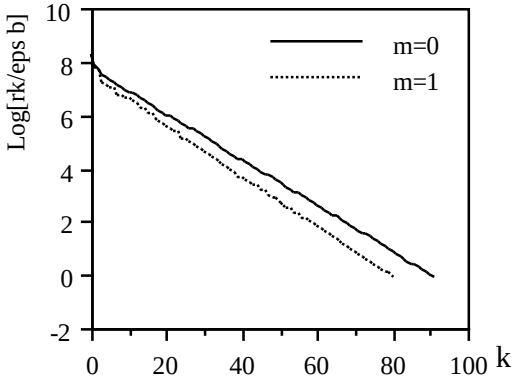


Figure 16: The norm of the logarithm of the residual vector $\mathbf{r}_k$ divided by the norm of **b** times ε, as a function of the iteration number, for n_{rel} = 2.5 + 1.4i (graphite).

Table 5 summarizes the results and gives the execution times of the PCGNR

n_{rel}	t_{iter}		number of iterations		t_{total}	
	m = 0	m = 1	m = 0	m = 1	m = 0	m = 1
1.05	6.54	13.3	6	3	39.2	40.0
1.5	6.54	13.3	32	16	209.3	213.2
1.33 + 0.05 i	6.54	13.3	20	10	130.8	132.8
1.7 + 0.1 i	6.54	13.3	52	45	34.0	59.8
2.5 + 1.4 i	6.54	13.3	91	80	59.5	106.3

Table 5: Total number of iterations and execution time per iteration (t_{iter}) and the total execution time (t_{total}), as a function of the relative refractive index n_{rel}; m=0 means no preconditioning, m=1 is a first order von Neumann preconditioner.

6] SUMMARY AND DISCUSSION

Our aim is to simulate the scattering of (visible) light by biological cells (specifically human white bloodcells). For this we exploit the Coupled Dipole method (see section 2). This model gives rise to a very large system of linear equations. As was shown in section 2, just the size of this system forces us to use an iterative method to solve the equations. Furthermore, acceptable run times can only be achieved if the iterative solver converges fast, and if the calculations can be performed at a very high speed. The first demand led to the choice of the CG method, the second one to a parallel MIMD computer.

The CG method is almost always applied to banded (sparse) matrices, coming from e.g. finite element discretizations. For these type of problems the CG method is successfully parallelized (see e.g.[35], chapter 8), also together with (polynomial) preconditioners (see e.g. [40]). Application of the CG method for full matrices is less common. However, system of equations with full, but diagonally dominant matrices as coming from the Coupled Dipole method, or from other computational Electromagnetics techniques (see e.g. [41]) can be solved efficiently with a (preconditioned) CG method.

The choice for a parallel distributed memory computer, as opposed to a vector supercomputer, was made for the following reason. For realistic particles the size of the matrix becomes to large to fit in memory. For instance, a double precision complex matrix of dimension 10^5 requires ±150 Gbytes RAM. Fortunately, the CG method does not need the complete matrix in memory, and the matrix elements can be calculated when needed. However, this calculation of matrix elements will be a severe bottleneck for the vector computer (cannot be vectorized), whereas this calculation of matrix elements parallelizes perfectly.

Parallelization of the CG method was done from scratch. First an abstract model of the computing system was defined. The parallel computing system is viewed as a set of communicating sequential processes, programmed in a number of consecutive cycles. During each cycle all processes perform work on local data, after which they synchronize and exchange data. With this abstraction the execution time of a parallel program can be expressed as equation 11. The first term is known and gives the smallest possible execution time for the parallel implementation (assuming an ideal sequential implementation). The other two terms are sources of efficiency reduction and must be minimized.

Our approach to parallelization was to start as general as possible, that is without any

specific knowledge of the computing system, and try to minimize T_{par} . Next limited characteristics of the computing system are specified, trying to find a good parallel implementation without detailed knowledge of the parallel computer. Finally numerical values of the model parameters (which of course strongly depend on the particular parallel computer) are needed to make an ultimate choice, if necessary, between different parallel implementations, and to calculate theoretical performance figures. This allows a generic approach to the parallelization of the application.

The decomposition is the only property of a parallel application that is independent of the computing system. The decomposition, together with the algorithm, lay down which calculations must be performed during a cycle, and which data movements must be performed after the calculations. Three data decompositions were considered: the rowblock -, columnblock -, and grid decomposition. The resulting expressions for T_{par} contain the only parameter of the abstract model of the computing system that does not explicitly refer to the parallel nature of the computing system, namely τ_{calc}. Using these expressions we showed that an implementation of the CG method, for complex symmetric matrices, with columnblock decomposition is always slower than with rowblock decomposition. This conclusions holds for all parallel computers which fall in the range of the abstract computing system. Although the difference in execution time will be small in practice, it is an important point of principle that these conclusions can be drawn at such a high level of abstraction.

The expressions for T_{par} contain terms for communication routines, and associated calculations. These terms can be expressed as a function of the parameters $\tau_{startup}$, τ_{comm}, and τ_{calc} if the connections between the parallel processors are specified. This means that a very important property of the parallel computer must be defined, the network topology. Here we restricted ourselves to the possibilities of the single transputer processing node, although the same analysis could have been made for more complex topologies such hypercubes (see [35]) or three dimensional meshes.

We analyzed the rowblock decomposition in combination with a ring topology and the grid decomposition in combination with a cylinder topology. Expressions of T_{par} as a function of the decompostion, topology and time constants are derived. These very detailed expressions are compared by an order of magnitude analysis, where we assumed n>>p. The first striking feature is the sensitivity of the rowblock-ring combination to a load imbalance (see equations 38 and 40), which is totally absent in the grid-cylinder combination (equation 39). Actually, the loadimbalance in the grid-cylinder combination is $O(1)\tau_{calc}$, which is negligible to the $O(n)\tau_{calc}$ due to the calculations in the partial vector accumulate (see equation 29). The total number of communication startups is smaller for the grid-cylinder combination then the rowblock-ring combination, but the total amount of transmitted data is the same (O(n), see equations 41 and 42). If $\tau_{startup}$ and τ_{comm} have the same order of magnitude, the startup time for communications is negligible compared to the pure sending time. In that case the communication overhead for the cylinder, and the less rich ring topology have the same order of magnitude. These considerations show that the rowblock-ring combination and the grid-cylinder combination will have comparable execution times. As long as the time constants have the same order of magnitude, and n>>p, both parallel implementation will have an efficiency close to 1 (see equation 44).

Based on Fox's equation for the fractional communication overhead f_c (equation 45) a smaller communication time for the grid-cylinder combination is expected. As was already argued in section 3.3 the partial vector gather operation is the bottleneck. This communication routine, implemented as proposed in section 3.3 does not exploit the full dimensionality of the cylinder. We did not look for a better implementation of the partial vector gather operation, using since the expected efficiencies of the parallel CG method with this implementation are already very satisfactory.

Finally the time constants were measured and introduced in the expression for T_{par}. Figures 2 to 5 show that the rowblock-ring - and the grid-cylinder combination are almost indistinguishable if one looks at their execution time. This shows that even on a low connectivity network as a ring it is possible to implement a parallel CG method with a very high efficiency (see figure 6), comparable with implementations on a cylinder topology. Keeping this in mind we implemented the rowblock-ring combination, based on two practical considerations; programming effort and system resources.

Figures 9 to 11 and table 4 show that the agreement between the theoretical and the measured values of T_{par} are very good, provided that n/p is not to small. For small n and large p the errors between theory and experiment become very high. In this situation every processor has only a few rows in memory and in that case the single parameter τ_{calc} to describe the calculation times in a single processor is not very accurate. However, this is not a real problem, since we are mainly interested in the situation n/p >> 1, where the errors between theory and experiment are very small.

The experiments were performed with relatively small matrices, which could be kept in memory. If the problem size grows the matrix cannot be kept in memory and the matrix elements must be calculated every time they are needed. This means that the total calculation time in the matrix vector product will increase, but that the expressions for the communication times remain the same (the same vectors must be communicated). The term T_{seq} and the loadimbalance terms will increase with a constant factor, but their order of magnitude remains the same. Therefore, due to the extra floating point operations the execution time will increase, but the parallel CG method will still run at a very high efficiency.

The last experiment concerned the convergence properties of the CG algorithm, and the influence of one preconditioner on the convergence speed. As was noted in section 2.3 a good preconditioner for a parallel PCGNR implementation must not only decrease the total number of floating point operations needed to find a solution, but furthermore the preconditioning steps must allow efficient parallelization. The polynomial preconditioner of equation 7 can be implemented using matrix vector products only (assuming that the coefficients γ_i are known at forehand). From section 3 it is obvious that the matrix vector product can be parallelized very efficiently in the rowblock-ring combination. Therefore the polynomial preconditioner is an ideal candidate for parallel implementations. Our implementation of the first order von Neuman preconditioner (m=1, see equation 8) supports this point. Measurements of the execution times for this implementation show that the efficiency is very close to 1 (data not shown). The same will be true for higher order preconditioners.

Figures 12 to 16 show that $|\mathbf{r}_k|$, the norm of the residual vector after the k'th iteration, decreases exponentially after every iteration. Even for graphite, with a large refractive index, the number of iterations needed for convergence is only 91, which is 2.3% of the matrix dimension n. For smaller refractive indices the number of iterations for m=1 is half of the number of iterations for m=0. The time for one iteration however is approximately a factor two higher (four instead of two matrix vector products per iteration). Therefore the total execution time is approximately the same (see table 5). For higher refractive indices the number of iterations is still decreased by the preconditioner, but not by a factor of two. In that case the total execution time for m=1 is much larger than for m=0. The first order von Neumann preconditioner is to crude to be an effective preconditioner for our type of matrices. In the future we will experiment with higher order von Neuman preconditioners, and other types of polynomial preconditioners. Especially if the size of matrix grows we expect that good preconditioning will be inevitable to obtain realistic execution times.

7] CONCLUSIONS

The preconditioned Conjugate Gradient method, for dense symmetric complex matrices, using polynomial preconditioners, can be parallelized successfully for distributed memory computers. Both a theoretical analysis of the time complexity and actual implementations prove this conclusion. Furthermore, the time complexity analysis shows that a parallel implementation on a one dimensional (ring) topology is as good as an implementation on a more complex two dimensional (cylinder) topology.

Theoretical predictions of the execution time of the parallel implementation agree very well with the experimental results. The abstract model of the computing system is well suited to describe the behaviour of regular numerical applications, such as the Conjugate Gradient method, on parallel distributed memory computers.

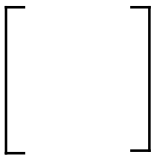
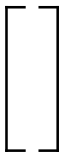
Convergence of the PCNR method, for some typical matrices, is very good. The first order von Neuman preconditioner, used as a test case, parallelizes very well; the efficiency of the implementation remains close to 1. However, the total execution time is not decreased. More research to better polynomial preconditioners is required.

ACKNOWLEDGEMENTS

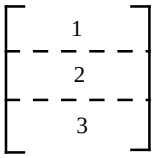
We wish to thank M.J. de Haan and S.G. Meijns for their contribution to the implementation of the Occam code and the performance measurements. One of us (AgH) wishes to acknowledge financial support from the Netherlands Organization for Scientific Research, grant number NWO 810-410-04 1.

APPENDIX A symbolic notation to visualize data decomposition

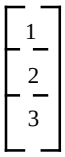
A matrix and vector are denoted by a set of two brackets:

matrix:  ,and vector:  .

The decomposition is symbolized by dashed lines in the matrix and vector symbols, as in

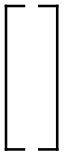


and



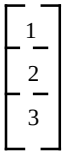
The matrix and vectors are divided in equal parts and distributed among the processing elements. The decomposition is drawn for three processing elements, but is extended to p processing elements in a straightforward way. The decomposition of the matrix is static, it remains the same during the computation. The decomposition of the vectors can take three distinct forms, depending on the calculation that was or has to be performed:

1] The vector is known in every processing element:

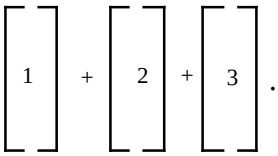


a special case is a scalar known in every processing element, which is depicted by [ ] ;

2] Parts of the vector are distributed among processing elements:

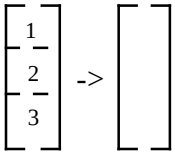


3] Every processing element contains a vector, which summed together gives the original vector. This "decomposition" is referred to as partial sum decomposition. This partial sum decomposition usually is the result of a parallel matrix vector product;



A special case is a partial sum decomposition of scalars, which is the result of parallel innerproducts. This will be depicted by [1] + [2] + [3]. Furthermore a mix of (2) and (3) is possible.

The '->' denotes a communication, e.g.



represents a vector gather operation, that is every processing element sends its part of the vector

to all other processing elements. The '=>' denotes a calculation, e.g.

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} => \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

is a parallel vector addition.

REFERENCES

[1] P.M.A. Sloot ElasticLight Scattering fromLeukocytes in the Development of Computer Assisted Cell Separation, Ph.D. dissertation, University of Amsterdam, 1988.

[2] A.W.S. Richie, R.A. Gray, and H.S. Micklem, Right Angle Light Scatter: a Necessary Parameter in Flow Cytofluorimetric Analysis of Human Peripheral Blood Mononuclear Cells, Journal of Immunological Methods **64** (1983) 109-117.

[3] J.W.M. Visser, G.J. Engh, and D.W. van Bekkum, Light Scattering Properties of Murine Hemopoietic Cells, Blood Cells **6** (1980) 391-407.

[4] M. Kerker, Elastic and Inelastic Light Scattering in Flow Cytometry, Cytometry **4** (1983) 1-10.

[5] M.C. Benson, D.C. McDougal, and D.S. Coffey, The Application of Perpendicular and Forward Light Scatter to Access Nuclear and Cellular Morphology, Cytometry **5** (1984) 515-522.

[6] M. Kerker, The Scattering of Light, (Acad. Press. , New York and London, 1969).

[7] H. van de Hulst, Light Scattering by Small Particles, (John Wiley & Sons , Dover, New York, 1981).

[8] C.F. Bohren and D.R. Huffman, Absorption and Scattering ofLight by Small Particles, (John Wiley & Sons , 1983).

[9] D.W. Shuerman, Light Scattering by Iregularly Shaped Particles, (Plenum Press , New York and London, 1980).

[10] E.M. Purcell and C.R. Pennypacker, Scattering and Absorption of Light by Nonspherical Dielectric Grains, The Astrophysical Journal **186** (1973) 705-714.

[11] R.W. Hart and E.P. Gray , Determination of Particle Structure from Light Scattering, J. Appl. Phys. **35** (1964) 1408-1415.

[12] P.M.A. Sloot, A.G. Hoekstra, H. Liet, van der , and C.G. Figdor, Scattering Matrix Elements of Biological Particles Measured in a Flow Through System:Theory and Practice, Applied Optics **28** (1989) 1752-17620.

[13] J. Hage, The Optics of Porous Particles and the Nature of Comets, Ph.D. dissertation, State univerisity of Leiden, 1991.

[14] D.E. Hirleman (Ed). Proceedings of the 2nd International Congress on Optical Particle Sizing. Arizona State University Printing services, 1990.

[15] W.J. Wiscombe and A. Mugnai, Scattering from Nonspherical Chebyshev Particles.2: Means of Angular Scattering Patterns, Applied Optics **27** (1988) 2405-2421.

[16] A. Mugnai and W.J. Wiscombe, Scattering from Nonspherical Chebyshev Particles. 3: Variability in Angular Scattering Patterns, Applied Optics **28** (1989) 3061-3073.

[17] J.D. Jackson, Classical Electrodynamics, (Jonh Wiley and Sons , New York, Chichester, Brisbane, Toronto and Singapore, 2nd ed., 1975).

[18] P.M.A. Sloot, M.J. Carels, and A.G. Hoekstra, Computer Assisted Centrifugal Elutriation, In Computing Science in the Netherlands , 1989).

[19] P.M.A. Sloot and C.G. Figdor, Elastic Light Scattering from Nucleated Blood Cells: Rapid Numerical Analysis, Applied Optics **25** (1986) 3559-3565.

[20] P.M.A. Sloot, A.G. Hoekstra, and C.G. Figdor, Osmotic Response of Lymphocytes Measured by Means of Forward Light Scattering: Theoretical Considerations, Cytometry **9** (1988) 636-641.

[21] A.G. Hoekstra, J.A. Aten, and P.M.A. Sloot, Effect of Aniosmotic Media on the T-lymphocyte Nucleus, Biophysical Journal **59** (1991) 765-774.

[22] P.M.A. Sloot, M.J. Carels, P. Tensen, and C.G. Figdor, Computer Assisted Centrifugal Elutriation.I. Detection System and Data Acquisition Equipment, Computer Methods and Programs in Biomedicine **24** (1987) 179-188.

[23] P.M.A. Sloot and A.G. Hoekstra, Arbitrarily Shaped Particles Measured inFlow Through Systems, In D.E. Hirleman, ed. Proceedings of the 2nd international congress on optical particle sizing , 1990).

[24] B.G. Grooth, L.W.M.M. Terstappen, G.J. Puppels, and J. Greve, Light Scattering Polarization Measurements as a New Parameter in Flow Cytometry, Cytometry **8** (1987) 539-544.
[25] G.H. Golub and Loan, Charles F., Matrix Computations second edition, (The John Hopkins University Press , Baltimore and London, 1989).
[26] B.T. Draine, The Discrete Dipole Approximation and its Application to Interstellar Graphite Grains, Astrophys. J. **333** (1988) 848-872.
[27] M.R. Hestenes and E. Stiefel, Methods of Conjugate Gradients for Solving Linear Systems, Nat. Bur. Standards J. Res. **49** (1952) 409-436.
[28] S.F. Ashby, T.A. Manteuffel, and P.E. Saylor, A Taxonomy for Conjugate Gradient Methods, Siam J. Numer. Anal. **27** (1990) 1542-1568.
[29] J.A. Meijerink and H.A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comp. **31** (1977) 148-162.
[30] O.G. Johson, C.A. Micchelli, and G. Paul, Polynomial Preconditioners for Conjugate Gradient Calculations, Siam J. Numer. Anal. **20** (1983) 362-376.
[31] Y. Saad, Practical Use of Polynomial Preconditionings for the Conjugate Gradient Method, Siam J. Sci. Stat. Comput. **6** (1985) 865-881.
[32] S.F. Ashby, Minimax Polynomial Preconditioning for Hermintian Linear Systems, Siam J. Matrix Anal. Appl. **12** (1991) 766-789.
[33] C. Tong, The Preconditioned Conjugate Gradient Method on the Connection Machine, International Journal of High Speed Computing **1** (1989) 263-288.
[34] S.F. Ashby, T.A. Manteuffel, and J.S. Otto, A Comparison of Adaptive Chebyshev andLeast Squares Polynomial Preconditioning for Hermitian Positive DefiniteLinear Systems, Siam J. Stat. Comput. **13** (1992) 1-29.
[35] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors, Volume 1, General Techniques and Regular Problems, (Prentice-Hall International Editions , 1988).
[36] C.A.R. Hoare, Communicating Sequential Processes, Communications of the ACM **21** (1978) 666-677.
[37] A. Basu, S. Srinivas, K.G. Kumar, and A. Paulray, A Model for Performance Prediction of Message Passing Multiprocessors Achieving Concurrency by Domain Decomposition, proceedings CONPAR '90, 1990 pp 75-85.
[38] W. Hoffmann and K. Potma Experiments with BasicLinear Algebra Algorithms on a Meiko Computing Surface, Tech. Rept. University of Amserdam, faculty of Mathematics and Computer Science, 1990.
[39] Occam® 2 Reference Manual, (Prentice Hall , 1988).
[40] M.A. Baker, K.C. Bowler, and R.D. Kenway, MIMD Implementation of Linear Solvers for Oil Reservoir Simulation, Parallel Computing **16** (1990) 313-334.
[41] J.I. Hage, M. Greenberg, and R.T. Wang, Scattering from Arbitrarily Shaped Particles: Theory and Experiment, Applied Optics **30** (1991) 1141-1152.