Parallel Execution Time Analysis for Least Squares Problems on Distributed Memory Architectures *

Laurence Tianruo Yang†‡ *Richard P. Brent* ‡

†*Department of Computer Science, St. Francis Xavier University*
P.O. Box 5000, Antigonish, B2G 2W5, Nova Scotia, Canada
‡*Computing Labortory, Oxford University*
Wolfson Building, Park Road, Oxford OX1 3QD, UK

Abstract

In this paper we study the parallelization of PCGLS, a basic iterative method which main idea is to organize the computation of conjugate gradient method with preconditioner applied to normal equations. Two important schemes are discussed. What is the best possible data distribution and which communication network topology is most suitable for solving least squares problems on massively parallel distributed memory computers. A theoretical model of data distribution and communication phases is presented which allows us to give a detail execution time complexity analysis and to investigate its usefulness. It is shown that the implementation of PCGLS, with a row-block decomposition of the coefficient matrix, on a ring of communication structure is the most efficient choice. Performance tests of the developed parallel PCGLS algorithm have been carried out on the massively distributed memory system Parsytec and experimental timing results are compared with the theoretical execution time complexity analysis.

1 Introduction

Many scientific and engineering applications such as linear programming [5], augmented Lagrangian methods for CFD [12], the natural factor method in partial differential equations [3, 14], tomography [30], seismic modeling [29], control theory [23], mechanical system and signal analysis [24], robotics [25], and structural analysis give rise to the least squares problems with a large and sparse coefficient matrix A. Excellent surveys of work in this area are [4, 11]. Minimizing by solving the normal equations is a common and often efficient approach, because $A^T A$ is symmetric and positive definite and it can be solved by using conjugate gradient method with preconditioner which was developed in the early 1950's by Hestenes and Stiefel [15]. The resulting method, PCGLS, is used as the basic iterative method to solve the least squares problems. Although this method does not explicitly form the normal equations, the rate of convergence still relies on the spectrum of $A^T A$. Preconditioning techniques that accelerate the convergence of these methods have received extensive attention in the literature [1, 22].

In many situations, especially when matrix operations are well-structured, these operations are suitable for implementation on vector and share memory parallel computers [9, 10]. But for parallel distributed memory machines the picture is entirely different. In general the matrices and the vectors are distributed over the processors, so that even when the matrix operations can be implemented efficiently by parallel operations, we can not avoid the communication required for information exchange. About discussions on how to reduce the communication costs on distributed memory systems, see [6, 7, 8, 20]. In a word, these communication costs become

*The author's email is lyang@stfx.ca

relatively more and more important when the number of the parallel processors is increased and thus they have the potential to affect the scalability of the algorithm in a negative way [7].

We are also highly concerned with preconditioners for massively parallel distributed memory computers because the preconditioning part is often the most problematic part in a parallel environment. The preconditioning techniques have been studied by many authors. In our paper we will focus on the Incomplete Modified Gram-Schmidt (IMGS) preconditioner, a class of incomplete QR factorization preconditioners, because their existences can be guaranteed [26].

Here, we present the functional aspects of parallelizing the PCGLS method for solving least squares problems. Parallelism is realized by decomposition of the problem. This can be a task decomposition or a data decomposition [13]. In the first case the problem is divided into a number of tasks that can be executed in parallel, and in the second case the data are divided into groups and the work on these grains is performed in parallel. The parallel parts of the problem are assigned to processing elements. In most case the parallel parts must exchange information. This implies that the processing elements are connected in a network. Parallelizing a problem basically implies answering the following questions: what is the best decomposition for problem or data distribution and which communication scheme is best suited.

A suitable metric is needed to decide which data distribution and communication scheme should be selected. We will follow the definition and the abstract model of the parallel computing system proposed in [17, 18]. We also choose the total execution time of the parallel program T_{par} as our metric in [17, 18]. The total execution time T_{par} depends on two variables and system parameters

$$T_{par} \equiv T_{par}(p, n, system\ parameters),$$

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

In general 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. Now we restrict ourselves first that our parallel computing system is Hoare's Communicating Sequential Processes [16]. The platform Parsytec's communication model suits our assumption well. Furthermore, we assume that the action within the sequential processes mainly consists of manipulating floating point numbers, and that the time needed to perform one floating point operation can be described with a single parameter τ_{calc}. So based on above assumptions, T_{par} can be specified as follows:

$$T_{par} \equiv T_{par}(p, n, \tau_{calc}, \tau_{startup}, \tau_{comm}, topology).$$

Finally it is assumed the parallel program consists of a sequence of cycles [2]. During a cycle all processors first perform some computations 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 the final assumption, T_{par} can be expressed as follows:

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

where

- $T_{seq} \equiv T_{seq}(p = 1, n)$, the execution time of the sequential algorithm, to be the execution time of the parallel algorithm on 1 processor.
- $T_{calc.np} \equiv T_{calc.np}(p, n, \tau_{calc}, topology)$, all calculation that can not be performed completely in parallel.
- $T_{comm} \equiv T_{comm}(p, n, \tau_{startup}, \tau_{comm}, topology)$, the total communication time of all cycles of the parallel program.

In this paper, we will focus on how to minimize T_{par} as a function of p and n and decide the corresponding data distribution and communication network topology combination for PCGLS method. Here we assume throughout that no preconditioner is used to verify the effect different parallelization strategies.

The left part of paper will be organized as follows. In section 2, we will present shortly our introduction for the PCGLS method with IMGS preconditioner. And an abstract model for data distribution or decomposition and communication schemes will be analyzed theoretically based on [17, 18, 28] in section 3. The comparison of actual implementation and theoretical total execution time complexity analysis will be described based on [17, 18, 27, 28].

2 PCGLS with IMGS Preconditioner

PCGLS with preconditioner M states as follows:

ALGORITHM PCGLS.

Let x_0 be an initial approximation, set

$$r_0 = b - Ax_0, \quad s_0 = p_0 = R^{-T}(A^T r_0), \quad \gamma_0 = (s_0, s_0);$$

for $k = 1, 2, \ldots$ compute

$$\begin{aligned}
&t_k = R^{-1} p_k;\\
&q_k = A t_k;\\
&\alpha_k = \gamma_k/(q_k, q_k);\\
&x_{k+1} = x_k + \alpha_k t_k;\\
&r_{k+1} = r_k - \alpha_k q_k;\\
&\theta_{k+1} = A^T r_{k+1};\\
&s_{k+1} = R^{-T}\theta_{k+1};\\
&\gamma_{k+1} = (s_{k+1}, s_{k+1});\\
&\beta_k = \gamma_{k+1}/\gamma_k;\\
&p_{k+1} = s_{k+1} + \beta_k p_k;
\end{aligned}$$

The PCGLS algorithm contains three distinct computational tasks per iteration

- Four matrix vector products, At_k, $A^T r_{k+1}$, $R^{-1}p_k$ and $R^{-T}\theta_{k+1}$, where $T_{seq}^{m-v} = 8n\tau_{calc}$.
- Two inner products, (q_k, q_k) and (s_{k+1}, s_{k+1}), where $T_{seq}^{inn} = (8n-2)\tau_{calc}$.
- Three vector updates, x_{k+1}, r_{k+1} and p_{k+1}, where $T_{seq}^{v-u} = (8n^2 - 2n)\tau_{calc}$.

Let $P_n = \{(i,j) | i \neq j, \quad 1 \leq i, j \leq n\}$ and assume that the matrix A has full column. Suppose we are given a set of index pairs P such that $P \in P_n$ and $(i,j) \in P$ implies that $i \leq i, j \leq n$. The set P determines which elements of the matrix R will not be retained in the approximate factorization, i.e., P is the drop set. Given these assumptions, we describe the Incomplete Modified Gram-Schmidt (IMGS) preconditioning technique fully as follows.

ALGORITHM Incomplete Modified Gram-Schmidt

$$\text{for} \quad k = 1, 2, \ldots, n \quad \text{do}$$

$r_{kk} = \|a_k^{(k)}\|_2;$
$q_k = a_k^{(k)}/r_{kk};$
for $j = k+1, \ldots, n$ do
$r_{k,j} = \begin{cases} 0 & (i,j) \in P \\ (q_k, a_j^{(k)}) & \text{otherwise} \end{cases}$
$a_j^{(k+1)} = a_j^{(k)} - r_{kj} q_k;$
endfor
end

Here a drop tolerance for elements P in R suggested by Jennings and Ajiz [19] is used, where the magnitude of each off-diagonal elements r_{ij} is compared against a drop tolerance τ scaled by the norm of the corresponding row norm $\| a_j \|_2$, i.e., $|r_{k,j}| < \tau d_j$, where $\tau = 0.02$ is recommended and

$$A = (a_1, a_2, a_3, \ldots, a_n)^T, \quad d_i = \| a_j \|_2. \tag{1}$$

For the fixed parameter selecting pattern, besides the general computation which we need $\frac{1}{2}(n^2+5n)$ inner products at most, here we also need to compute n inner products for the drop tolerance parameter $d_j = \| a_j \|_2$, and if $(i,j) \in P$ we even do not need to compute the inner products. So combining with the above situations, we use the average number of them as the estimated number of the inner products in IMGS preconditioning procedure. It also shows clearly that the choice of P, not surprisingly, can be crucial not only to the quality of the preconditioner but also to the combined efficiency of the parallel implementation.

The IMGS preconditioning procedure requires approximately $\frac{1}{4}(n^2 + 5n)$ inner products, and $\frac{1}{2}(n^2 + n)$ vector updates, and n matrix-vector multiplications. And a single iteration of PCGLS requires two inner products, and three vector updates and four matrix-vector multiplications. So, the total execution time per iteration is

$$T_{seq}(n) = (14n^3 + \frac{87}{2}n^2 + \frac{59}{2}n - 4)\tau_{calc}, \tag{2}$$

where we ignore the two divisions α_k and β_k and the time for initialization if the number of iteration is large enough.

3 Data Distribution and Communication Schemes

Based on [27, 28], three different data decompositions of the matrix will be discussed: row-block decomposition, column-block decomposition and grid decomposition. We will demonstrate in the paper that, theoretically and practically, the row-block decomposition is much more efficient then the column and grid decompositions.

First we will present the basic parallel vector operations and their execution time complexity mainly based on [17, 18].

3.1 Vector update operations

The parallel vector update operations can be performed in parallel. The computing time for the parallel vector update operations [17, 18, 27, 28]:

$$T_{par}^{v-u}(p,n) = \frac{T_{seq}^{v-u}(n)}{p} + 8(\lceil \frac{n}{p} \rceil - \frac{n}{p})\tau_{calc}, \tag{3}$$

where $\lceil x \rceil$ is the ceiling function of x. It does not have to communicate data, so $T_{comm} = 0$. And non-parallel part is a pure load balancing effect. From (3), we can know that:

$$T^{v-u}_{np.calc}(p,n) = 8(\lceil \frac{n}{p} \rceil - \frac{n}{p})\tau_{calc}, \tag{4}$$

if the length of the vector is not divisible by the number of processor p, then some processors receive pieces of the vector with size $\lceil \frac{n}{p} \rceil$, and some with size $\lceil \frac{n}{p} \rceil - 1$. This will lead to a load imbalance, of which the effect is presented by (4). In general, for large grain size, $n/pT_{np.calc}$ is very small compared with T_{seq}/p.

3.2 Inner product operations

The total computing time for the parallel inner production operations is [17, 18, 27, 28]:

$$T^{inn}_{par}(p,n) = \frac{T^{inn}_{seq}(n)}{p} + (t^{inn}_{ca.calc} - 2(1 - \frac{1}{p})\tau_{calc}) + 8(\lceil \frac{n}{p} \rceil - \frac{n}{p})\tau_{calc} + t_{ca},$$

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 processors. The $t_{ca.np}$ is a computing time introduced by summing the partial inner products after or during the scalar accumulation. It is quite easy to know that:

$$T^{inn}_{comm}(p,n) = t_{ca}, \quad T^{inn}_{calc.np} = (t^{inn}_{ca.calc} - 2(1 - \frac{1}{p})\tau_{calc}) + 8(\lceil \frac{n}{p} \rceil - \frac{n}{p})\tau_{calc}.$$

There are two terms included in this part. The first term describes the summations in the parallel vector inner product that can not be performed completely in parallel and The second term is the load imbalance term.

3.3 Matrix vector multiplication operations

The execution times for vector updates and inner products are independent of the decomposition of the coefficient matrix. But the matrix vector multiplication operations are the only matrix dependent parts of the algorithm. So in next subsection, we will examine these operations in these three different decompositions mainly based on [17, 18].

3.4 Decomposition models

From PCGLS algorithm, we can see clearly that it consists of the following three types of data: scalars, vectors and matrices. All operations are performed on vectors and scalars. Therefore it is quite natural to define a static decomposition of the matrices. This prevents that large pieces of the matrices have to be sent over the network. We will present the matrix vector product for the following three situations based on the analysis on [17, 18, 27, 28]: row-block decomposition, column-block decomposition and grid decomposition.

3.4.1 The row-block decomposition

The row-block decomposition is achieved by dividing A into blocks of rows, with every block containing $\lceil \frac{N}{p} \rceil$ or $\lceil \frac{N}{p} \rceil - 1$ consecutive rows, and assigning one block to every processing element. The total execution time for this operation is [17, 18, 27, 28]:

$$T^{m-v-row}_{par}(p,n) = \frac{T^{m-v}_{seq}}{p}(n) + (8n - 2)(\lceil \frac{n}{p} \rceil - \frac{n}{p})\tau_{calc} + t_{v-g},$$

where t_{v-g} 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.

3.4.2 The column-block decomposition

The column-block decomposition is achieved by grouping $\lceil \frac{N}{p} \rceil$ or $\lceil \frac{N}{p} \rceil - 1$ consecutive columns of A into blocks and distributing these blocks among the processing elements. The total execution time for this operation is [17, 18, 27, 28]:

$$T_{par}^{m-v-col} = \frac{T_{seq}^{m-v}(n)}{p} + (t_{va.calc} - 2n(1 - \frac{1}{p}\tau_{calc})) + 8n(\lceil \frac{n}{p} \rceil - \frac{n}{p})\tau_{calc} + t_{va},$$

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 the non-parallel part consists of a load balancing term and a term for the summations in the partial vector gather operation. T_{comm} is the communication time for the partial vector accumulate operation.

3.4.3 The grid decomposition

The matrix is decomposed in p blocks, which are distributed among the processing elements. The matrix A^T also is decomposed as a grid. The total execution time for this parallel matrix vector product can be expressed as follows [17, 18, 27, 28]:

$$T_{par}^{m-v-grid} = \frac{T_{seq}^{m-v}}{p}(n) + (t_{pva.calc} - 2n(\lceil \frac{n}{\sqrt{p}} \rceil \frac{1}{n} - \frac{1}{p})\tau_{calc}) + 8(\lceil \frac{n}{\sqrt{p}} \rceil^2 - \frac{n^2}{p})\tau_{calc} + t_{p-g} + t_{pva},$$

where t_{p-g} 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. The non-parallel part consists of a load balancing term and float-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.

The complete comparison of the execution time for these three different matrix decompositions is presented fully in [27, 28]. Before we present the theoretical results, we should define this term $T_{loss}(p, N)$ as follows:

$$T_{loss}(p, N) = p(T_{calc.np}(p, N) + T_{comm}(p, N)). \tag{5}$$

From the theoretical expressions, it is easy to know that if communication times can be neglected the row-block decomposition has the smallest T_{loss}. The term T_{loss} is only introduced by evaluating the partial scalar sum decompositions of the inner products. The grid and column decompositions also give rise to vector partial sum decompositions in the parallel matrix vector products. This evaluation adds up to T_{loss}.

It also can be shown that T_{loss} of the column-block decomposition is always bigger than that of the row-block decomposition. For every interconnection scheme of the processing elements the vector accumulate plus scatter operation of the column block decomposition [17, 18] involves messages that are bigger in size than, at least equal to the messages in the vector gather operation of the row-block decomposition. So, as proved in [17, 18, 27, 28], we can get the following relations:

$$T_{comm}^{column} \geq T_{comm}^{row}, \qquad T_{calc.overhead}^{column} > T_{calc.overhead}^{row},$$

so that we conclude that

$$T_{loss}^{column} > T_{loss}^{row}. \tag{6}$$

Similar comparison for matrix vector multiplication, we can conclude that the row-block decomposition is much more efficient than the column and grid decompositions of A for any parallel computers.

4 Theoretical Time Complexity Analysis

We will present the execution time complexity analysis of the row-block decomposition on a ring [17, 18, 27, 28]. We also will describe the row-block decomposition on a binary tree and a torus based on [27] which give a fully theoretical analysis. The total execution time on the torus is higher than on the ring and binary tree topologies. The last two are comparable. Furthermore our presentation will focus on the grid decomposition on a torus topology.

4.1 The row-block decomposition on ring topology

If we assume that point to point communication of n floating points takes a time

$$t_{point-to-point} = \tau_{startup} + n\tau_{comm}, \tag{7}$$

then $t_{v-g}(p, N)$, the time needed for the vector gather operation, can be expressed as follows [17, 18, 27, 28]:

$$t_{v-g}^{ring}(p, N) = \lfloor \frac{p}{2} \rfloor \tau_{startup} + \lfloor \frac{p}{2} \rfloor \frac{16N}{p} \tau_{comm}.$$

On ring topology, The expressions for t_{ca} and $t_{ca.np}$ in inner product operation can be described as follows [17, 18, 27, 28]:

$$t_{ca}^{ring} = \lfloor \frac{p}{2} \rfloor (\tau_{startup} + 16\tau_{comm}), \quad t_{ca.calc}^{ring} = 2(p-1)\tau_{calc}.$$

4.2 The grid decomposition on a torus topology

From the theoretical analysis in [27, 28], the execution time of the partial vector gather operation t_{p-g}^{torus} on the torus topology is [17, 18, 27, 28]:

$$t_{p-g}^{torus} = (\sqrt{p} - 1)(\tau_{startup} + 16\lfloor \frac{n}{p} \rfloor \tau_{comm}).$$

The execution time of the vector accumulate operation t_{pva}^{torus} is [17, 18, 27, 28]:

$$t_{pva}^{torus} = \lfloor \frac{\sqrt{p}}{2} \rfloor (\tau_{startup} + 16\lceil \frac{n}{\sqrt{p}} \rceil \tau_{comm}) + (\sqrt{p} - 1)(\tau_{startup} + 16\lceil \frac{n}{p} \rceil \tau_{comm}),$$

and $t_{pva.calc}^{torus}$, the time to evaluate the partial sums after or during the vector accumulation can be expressed as follows [17, 18, 27, 28]:

$$t_{pva.calc}^{torus} = 2(\sqrt{p} - 1)\lceil \frac{n}{\sqrt{p}} \rceil \tau_{calc}.$$

On torus topology, the complex accumulate t_{ca}^{torus} is quite simple like [17, 18, 27, 28]:

$$t_{ca}^{torus} = \lfloor \frac{\sqrt{p}}{2} \rfloor (\tau_{startup} + 16\tau_{comm}) + (\sqrt{p} - 1)(\tau_{startup} + 16\tau_{comm}), \quad t_{ca.calc}^{torus} = 4(\sqrt{p} - 1)\tau_{calc}.$$

The derivation of the expression for the communication and non-parallel routines, as a function of the system parameters, allows us to compare T_{par} of the PCGLS with IMGS preconditioner for two different parallelization strategies.

Here we will compare T_{np} and T_{comm} of both parallelization strategies in some limited cases. From the previous parts, we know that

$$T_{np}^{PCGLS.IMGS} = \frac{1}{4}(n^2+5n)T_{np}^{inn} + \frac{1}{2}(n^2+n)T_{np}^{v-u} + nT_{np}^{m-v}, \tag{8}$$

and

$$T_{comm}^{PCGLS.IMGS} = \frac{1}{4}(n^2+5n)T_{comm}^{inn} + \frac{1}{2}(n^2+n)T_{comm}^{v-u} + nT_{comm}^{m-v}. \tag{9}$$

Combining the expression of the previous section with (8) and (9), we get

$$(T_{np}^{PCGLS.IMGS})_{ring}^{row-block} = ((14n^2+12n)(\lceil\frac{n}{p}\rceil - \frac{n}{p}) + \frac{1}{2}(n^2+5n)(p+\frac{1}{p}-2))\tau_{calc}, \tag{10}$$

$$(T_{comm}^{PCGLS.IMGS})_{ring}^{row-block} = (\frac{1}{4}n^2+\frac{9}{4}n)\lfloor\frac{p}{2}\rfloor\tau_{startup} + 16\lfloor\frac{p}{2}\rfloor(n\lceil\frac{n}{p}\rceil + \frac{1}{4}n^2+\frac{5}{4}n)\tau_{comm}, \tag{11}$$

and

$$\begin{aligned}(T_{np}^{PCGLS.IMGS})_{torus}^{grid} &= n(\lceil\frac{n}{\sqrt{p}}\rceil^2 - \frac{n^2}{p})\tau_{calc} \\ + \quad &((6n^2+14n)(\lceil\frac{n}{p}\rceil - \frac{n}{p}) + \frac{1}{2}(n^2+5n)(2\sqrt{p}+\frac{1}{p}-3))\tau_{calc} + n((2\sqrt{p}-4)\lceil\frac{n}{\sqrt{p}}\rceil + 2\frac{n}{p})\tau_{calc},\end{aligned} \tag{12}$$

$$\begin{aligned}(T_{comm}^{PCGLS.IMGS})_{torus}^{grid} &= (\frac{1}{4}n^2+\frac{13}{4}n)(\sqrt{p}-1)\tau_{startup} + 16n\lfloor\frac{\sqrt{p}}{2}\rfloor\lceil\frac{n}{p}\rceil\tau_{comm} \\ + \quad &(32n(\sqrt{p}-1)\lceil\frac{n}{p}\rceil + 4(n^2+5n)(\lfloor\frac{\sqrt{p}}{2}\rfloor + \sqrt{p}-1))\tau_{comm} + ((\frac{1}{4}n^2+\frac{9}{4}n)\lfloor\frac{\sqrt{p}}{2}\rfloor)\tau_{startup}.\end{aligned} \tag{13}$$

If we assume that $n \gg p$ in case of a perfect load balance, we know from (10) and (12) that T_{np} and T_{comm} for the row-block-ring combination and the grid-torus combination are comparable.

As long as τ_{calc}, τ_{comm} and $\tau_{startup}$ have the same order of magnitude, and $n \gg p$, the efficiency of the parallel PCGLS with IMGS preconditioner can be very close to unity.

5 Numerical Experiments

The Parsytec architecture combines the state-of-the-art RISC processor PowerPC-601 and advanced transputer communication technology and achieves top-end supercomputer performance with a scalable architecture supporting thousands of processor and offers a clear future upgrade path to PowerPC-604 and PowerPC-620 microprocessors. About the test matrices, we use the matrices from the simulation of the Elastic Light Scattering from arbitrary shaped micron sized biological particles by using the Couple Dipole method described in [18, 21]. In this case, A is symmetric $3N$ by $3N$ complex matrices with $N \sim 10^5$ and we generate the right-hand side vector consistent with a solution vector whose components are equal to 1.

5.1 Performance model

In [27], several kinds of parallelism at three levels support by Parsytec architecture is described. Four types of communication and building some topologies are described as well. For hardware, the model parameters $\tau_{calc}, \tau_{startup}$ and τ_{comm} depend on the hardware, on runtime environment and on compilers. Direct derivation of these parameters from hardware specifications alone gives always too optimistic values. Therefore reliable

numerical values of the model parameters can only be obtained by actual measurements on the parallel computers. The corresponding values have been described in [27, 28].

We have computed T_{loss} as function of p and N. The experimental results give T_{loss} as a function of N, for p equals 32 and 64, both for the row-block-ring and the grid-torus combinations. For very small N the grid-torus has a smaller T_{loss}. However, if N grows the T_{loss} of the row-block-ring combination is always smaller than for the grid-torus combination. This was concluded in the previous section and also experimental results. Due to the limited space in the journal space, we will present the detailed results in somewhere else. For realistic problems the row-block decomposition implemented on a ring topology is the best choice for PCGLS with IMGS preconditioner.

Also it is preferable to choose the row-block-ring combination from an implementation point of view because it introduces just one communication routine, namely the gather operation. PCGLS with IMGS preconditioner via a grid-decomposition contains more and more complex communication routines. Furthermore, if you look at this issue as a user, the row-block-ring combination is also favorable due to its following 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. These above mentioned considerations are in favor of the row-block-ring combination. If the number of rows per processor is large enough the efficiency will be very close to unity.

5.2 Performance measurements

We have implemented the parallel PCGLS with IMGS preconditioner, with row-block decomposition of A, on a ring topology. These performance measurements are compared with the theoretical expectations for this derived in the previous section. The preliminary results of the measurement of $T_{par}(p, N)$, the time per iteration as a function of p and N, and comparison of the results with theory. The detailed description will be presented in the future due to the limited space here. It does gives the results for N equals 64, 250, 496, 918 and $1 \leq p \leq 64$. The relative difference between theory and measurement is almost 3.5%.

References

1. S. Ashby. *Polynomial preconditioning for conjugate gradient methods*. PhD thesis, Department of Computer Science, University of Illinois Urbana-Champaign, 1987.
2. A. Basu, S. Srinivas, K. G. Kumar, and A. Paulray. A model for performance prediction of message passing multiprocessors achieving concurrency by domain decomposition. In *Proceedings of CONPAR'90*, pages 75–85, 1990.
3. M. W. Berry and R. J. Plemmons. Algorithms and experiments for structural mechanics on high performance architecture. *Computer Methods in Applied Mechanics and Engineering*, 64:1987, 487-507.
4. Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1995.
5. I. C. Chio, C. L. Monma, and D. F. Shanno. Further development of a primal-dual interior point method. *ORSA Journal on Computing*, 2(4):304–311, 1990.
6. E. de Sturler. A parallel restructured version of GMRES(m). Technical Report 91-85, Delft University of Technology, Delft, The Netheland, 1991.
7. E. de Sturler. A parallel variant of the GMRES(m). In *Proceedings of the 13th IMACS World Congress on Computational and Applied Mathematics*. IMACS, Criterion Press, 1991.
8. E. de Sturler and H. A. van der Vorst. Reducing the effect of the global communication in GMRES(m) and CG on parallel distributed memory computers. Technical Report 832, Mathematical Institute, University of Utrecht, Utrecht, The Netheland, 1994.

9. J. W. Demmel, M. T. Heath, and H. A. van der Vorst. Parallel numerical algebra. *Acta Numerica*, 1993. Cambridge Press, New York.

10. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, PA, 1991.

11. I. S. Duff and J. K. Reid. A comparison of some methods for the solution of sparse over-determined system of linear equations. *Journal of the Institute of Mathematics and Its Applications*, 17(3):267–280, 1976.

12. M. Fortin and R. Glowinski. *Augmented Lagrangian Methods: Application to the Numerical Solution of Boundary-value Problems*. NH, 1983.

13. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. Solving problems on concurrent processors. In *General Techniques and Regular Problems.*, volume 1. Prentice-Hall International Editions, 1988.

14. M. T. Heath, R. J. Plemmons, and R. C. Ward. Sparse orthogonal schemes for structure optimization using the force method. *SIAM Journal on Scientific and Statistical Computing*, 5(3):514–532, 1984.

15. M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear system. *J. Res. Nat. Bur. Stds*, B49:409–436, 1952.

16. C. A. R. Hoare. Communicating sequential processes. *Communications of ACM*, 21(666-677), 1978.

17. A. G. Hoekstra, P. M. A. Sloot, M. J. de Haan, and L. O. Hertzberger. Time complexity analysis for distributed memory computers, implementation of a parallel conjugate gradient method. In J. van. Leeuwen, editor, *Computing Science in the Netherlands*, pages 249–266. Elsevier Science, 1991.

18. A. G. Hoekstra, P. M. A. Sloot, W. Hoffmann, and L. O. Hertzberger. Time complexity of a parallel conjugate gradient solver for light scattering simulations: theory and SPMD implementation. Technical report, Department of Computer System, University of Amsterdam, 1992.

19. A. Jennings and M. A. Ajiz. Incomplete methods for solving $A^T Ax = b$. *SIAM Journal on Scientific and Statistical Computing*, 5:978–987, 1984.

20. C. Pommerell. *Solution of large unsymmetric systems of linear equations*. PhD thesis, ETH, 1992.

21. E. M. Purcell and C. R. Pennypacker. Scattering and absorption of light by nonspherical dielectric grains. *The Astrophysical Journal*, 186:705–714, 1973.

22. Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing*, 6:865–881, 1985.

23. S. Sagara and K. Wada. On-line modified least squares parameter estimation of linear discrete dynamic systems. *International Journal on Control*, 25(3):329–343, 1977.

24. S. Van Huffel and J. Vandewalle. Iterative speed improvement for solving slowly varying total least squares problems. *Mechnical System and Signal Processing*, 2:327–348, 1988.

25. B. Walden. *Least sqaures methods and applications in robotics*. PhD thesis, Linköping University, Sweden, 1994.

26. T. Yang. Error analysis for incomplete modified Gram-Schmidt preconditioner. In Proceedings of *Prague Mathematical Conference (PMC-96)*, July 1996. Mathematical Institute of the Academy of Sciences, Zitza 25, CZ-115 67 Praha, Czech Republic.

27. T. Yang. Execution time analysis for least squares problems on massively parallel distributed memory computers. In Proceedings of *International Conference on Computational Modeling and Computing (CMCP-96)*, September 1996. Dubna, Russia.

28. T. Yang. Best data decomposition model of PCGLS for parallel sparse least squares problems. In Proceedings of *Second Workshop on Advanced Parallel Processing Technologies (APPT-97)*, May 1997. Koblenz, Germany.

29. T. Yang. Parallel Newton-GMRES method for large and sparse nonlinear equations in computational seismic modeling. In Proceedings of *The International Symposium of High Performance Computing in Seismic Modeling (HPCSM-97)*, June 1997. Zaragoza, Spain.

30. T. Yang. Solving sparse least squares problems in seismic travel tomography on bulk synchronous parallel architectures. In Proceedings of *The International Symposium of High Performance Computing in Seismic Modeling (HPCSM-97)*, June 1997. Zaragoza, Spain.