Block Size Selection of Parallel LU and QR on PVP-based and RISC-based Supercomputers*

Yunquan Zhang
Institute of Software, CAS
P.O. Box 8718
Beijing,100080, P.R. China
zyq@mail.rdcps.ac.cn

Ying Chen
Institute of Computing Technology, CAS
P.O. Box 2704
Beijing,100080, P.R. China
yingchen@ncic.ac.cn

Yuan Tang
Parallel Processing Institute, Fudan University
220 Handan Road
Shanghai, 200433, P.R. China
yuantang@fudan.edu.cn

ABSTRACT

In this paper, we proposed a unified framework and tried to address the optimal block size selection problem for parallel blocked LU and QR factorization algorithm used in ScaLAPACK package, since they use two dimensional block cyclic data distribution fashion [12], block size plays important role in determining the final performance. Through the analysis with our proposed framework and experiments on small scale system configuration, we found that among all factors that affect performance, load balance and local block size selection play key roles in determining the optimal block size on two different type parallel computing platforms: SR2201(PVP(Pseudo-Vector Processing) based MPP machine) and DAWNING 3000(RISC-based SMP cluster). In fact, the optimal parallel block size is determined by the processor grid shape and problem size. Based on this observation, optimal block size prediction formula for double precision real parallel blocked LU and QR on SR2201 and DAWNING3000 with processor grid shape and problem size as parameters were given, whose prediction results can match well with the large scale system configuration and large problem size experimental results. The application of our framework on other parallel machines and on other applications program wound be the future work.

Categories and Subject Descriptors

G.4 [**Mathematical Software**]: Parallel and vector implementations; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Performance

Keywords

Optimal Parallel Block Size, PVP, RISC, ScaLAPACK, LU, QR

1. INTRODUCTION

Blocking [8, 9, 10] and tiling [13, 18, 19, 20] have long been the effective loop transformation and optimization techniques used to reduce the work set size, to increase data reuse in cache, and to provide more resources for compiler optimization. Blocking and tiling are thus often used to exploit performance for processors with small and fast cache, large register files, pipeline and super-scalar function. However, the selection of optimal blocking parameters is still a difficult task until now. This problem becomes complicated since it is the interaction between program structure, memory access pattern and underlying memory hierarchy organization on which the program to be executed. To select optimal blocking parameters for one specific application program on one specific machines, the programmer needs to make detailed analysis on cache miss behavior [8, 10] of the application program, and to find out the block size that minimize the miss rate and cache conflicts. This is not a simple task for common user. Thus the common way to deal with such situation is still through lots of experiments to find the changing trend of local computational speed with block size variation.

The parallelization of blocked sequential algorithms makes this problem further complicated. Now this problem becomes the interaction between the program characteristics, underlying single processor memory hierarchy, interconnection network, memory and load balance requirement of parallelization. The optimal block size must be selected to maximize the data reuse in cache, to maximize the network bandwidth utilization while keeping memory and load balance among parallel processes. However, these goals usually can not be satisfied at the same time, the user needs to trade off among them, i.e., to satisfy the most important factor while not too worse on other factors.

As we know, the common way to deal with this problem is to model the execution time of a parallel blocked algorithm with block size as one parameter, and the optimal block size can thus be found out through minimizing the total execution time function [6, 14]. Although this is an effective way, it is too difficult to be used by common user since it requires

*This work was supported in partial by the National Natural Science Foundation of China under contract No.60303020 and No.60533020, the National 863 Plan of China under contract No.2006AA01A102 and No.2006AA01A125, and the Open Foundation No.200505 of State Key Laboratory of Networking and Switching Technology.

ATIP 3rd China HPC Workshop, November 11, 2007, Reno, NV

too much detailed knowledge on parallel blocked algorithm implementation [6], i.e., the user needs to model each subroutine in this parallel blocked implementation and to determine too many model parameters. The method used in [14] simplifies itself by simple assumption on local computational speed versus block size variation(the computational speed is constant when the block size changed), but this makes themselves reach impractical conclusions that block size 1 is the optimal block size and they provide no experimental results to validate their conclusions. Another way [16] to resolve this difficult task is to distribute data using small storage block size(b, keeping better load balance) while to perform parallel computation using larger algorithmic block size(w, higher local computational speed), the two important goals thus can be reached at the same time. However, this is at the cost of more communication volume and cost of block LU factorization(panel factorization) and a fully distributed version of parallel BLAS library which supports the distributed panel matrix operations over several processor rows and columns. The later requirement makes this method not applicable to machines without support of such parallel BLAS library. But this would be a useful idea and can be adopted in the future design of parallel numerical library.

In this paper, we have no intention to address the problem of single processor optimal block size selection since it is specific for certain application and machines, and many works have been done on this topic. The interested reader can refer to [8, 9, 10] for more discussions. We assume this problem can be resolved by heuristic method like "climbing hill" method and cache miss analysis [10]. Based on these assumptions, we want to provide one uniform framework on optimal or suboptimal block size selection problem of parallel program using blocked algorithm. The relative importance of framework constraint factors can be analyzed based on theoretical model and experimental results. Our purpose is to show that after a few small scale experiments on optimal block size selection and match these results with framework constraints, we can find out the most important constraint and its relationship with optimal block size selection. This relationship can be used on predicting optimal or suboptimal block size for larger amount of processors and larger problem size. The timing drivers of parallel double precision real LU(DLU for short) and QR(DQR for short) factorization in ScaLAPACK package will be used as such parallel applications. Using LU and QR as an example, we demonstrated that our framework can determine the most important constraint factor, i.e., load balance constraint, and finally the simple prediction formulas for the near optimal block size of parallel LU and QR on SR2201 and DAWNING 3000 were given respectively, which matched well with experimental results on large scale machine and problem size. The application of our framework on other parallel machines and on other applications program wound be the future work.

This paper is organized as follows: in section 2, related works on this topic were given with detailed discussion and comparisons. Our framework was proposed in section 3. In section 4, the parallel DLU and DQR factorization in ScaLAPACK were used as examples to demonstrate how to apply our framework on two different platforms, and detailed experimental results were compared with the prediction results of the near optimal block size prediction formula given by our framework. Two experimental platforms, HITACHI SR2201 and DAWNING 3000, were also introduced in this section. Finally conclusions were given in section 5.

2. RELATED WORK

To our knowledge, the problem on optimal block size selection of parallel blocked algorithm in ScaLAPACK package was first proposed by Blackford, etc. in [1]. It is stated in [1] that *The chosen block size impacts the amount of workspace needed on every process. This amount of workspace is typically large enough to contain a block of columns or a block of rows of the matrix operands. Therefore, the larger the block size, the greater the necessary workspace, i.e the smaller the largest solvable problem on a given grid of processes. For Level 3 BLAS blocked algorithms, the smallest possible block operands are of size $r \times c$. Therefore, it is good practice to choose the block size to be the problem size for which the BLAS matrix-multiply GEMM routine achieves 90% of its reachable peak. Determining optimal, or near optimal block sizes for different environments is a difficult task because it depends on many factors including the machine architecture, speeds of the different BLAS levels, the latency and bandwidth of message passing, the number of process available, the dimensions of the process grid, the dimension of the problem, and so on. However, there is enough evidence and expertise for automatically and accurately determining optimal, or near optimal block sizes via an enquiry routine. Furthermore, for small problem sizes it is also possible to determine if redistributing n^2 data items is an acceptable cost in terms of performance as well as memory usage. In the future, we hope to calculate the optimal block size via an enquiry routine.*

The framework proposed in this paper just wants to find out the way on how to unify these factors in determining the rules of parallel optimal block size selection.

In [14], the optimal data distribution problem of parallel LU factorization was discussed based on a computational model and a parameterized data distribution function. The parallel execution time function of LU was formed with the processor grid $r \times c$, block size $b_0 \times b_1$ as parameters. The optimal block size selection problem was thus transformed into a minimization problem on total parallel execution time. However, the paper reached impractical conclusion of optimal block size $b_0 = b_1 = 1$ since it assumes the local computation speed would not change with the block size. And the author gives no experiments to verify their conclusion.

The optimal block size selection problem was totally avoided in [16] through distinguishing the storage block(b) and algorithmic block(w) by **panel distribution** technique. The ideal behind panel distribution is to distribute data using storage block size b with b small enough to keep better load balance, while the algorithm proceeds using the algorithmic block size w with w large enough to reach high local computational speed. Thus parallel program can achieve load balance while still keeping high local computational speed at the expense of increased communication overhead, both in startup and volume cost. However, this requires too much work on redesigning the parallel BLAS library. This method is not general enough. But the panel distribution technique would be useful ideal for future design of parallel numerical linear algebra package on distributed memory machines.

Most recently, F. Desprez, etc., solved this optimal block size selection problem of parallel LU in ScaLAPACK [6].

Table 1: Parameters Used in this Paper

Parameter	Definition
$l_b \leq b_0 \leq u_b$	Range of near Optimal Local Computation Block Size
L_0	Communication Saturation Message Length Size
C	L1 Cache Size
l	L1 Cache Line Size
p	Number of Parallel Processors
m	Available Memory Size of Uniprocessor
n	Problem Dimension Size
b_{bal}	Load Balance Parallel Block Size
b_{opt}	Optimal Parallel Block Size
r	Number of processor rows in processor grid
c	Number of processor columns in processor grid
e	The byte size of data type. (8 for double precision)
k	The times that block cyclic data distribution wrapping around the processor grid column

They built the parallel execution time model of *LU* through detailed step by step cost analysis on each subroutine with many parameters that must be determined by experiments. Then derivative was performed on the time function in real domain with the sum of the derivative equals zero. The integer part of the positive real root of this equation is just the optimal block size to be selected. However, this method requires too many parameters to be determined and too much knowledge on the internal details of parallel *LU* in ScaLAPACK. On the contrary, our framework only needs stages description and its computation complexity of parallel algorithm.

In [3], the author discussed the implementation of parallel *LU* with analyzing on the computation complexity of three stages. The block size selection problem was also discussed using the execution time function given by the author. The similar conclusion was reached as ours, i.e., keeping $l = ck$ no less than a given value while selecting block size b as large as possible. But the author didn't give a formula for optimal block size selection.

3. FRAMEWORK

In this section, we will show our framework for optimal parallel block size selection problem of ScaLAPACK library based applications. We first define several parameters used in this framework, and then give the detailed description on our framework.

3.1 Parameters

The parameters used in this framework are given in Table 1.

3.2 Framework Description

In this section, we give the framework description as follows:

1. First, the user needs to find the optimal or suboptimal local computation block size or the range of performance acceptable block size. This can be solved through experiments on kernel computation subroutines. In ScaLAPACK, the optimal local block size selection is mainly for BLAS3 subroutines, especially for matrix multiplication. For processors with small and fast cache(RISC-based processor), the block size parameter can often be the same or multiple of cache line size l, i.e. $b_0 = gl$, where g is a constant, DAWNING 3000 is such a RISC-based MPP machine; For processors with large register files and vector function(Vector-based processor) which can combine loop unrolling [2] with blocking, the block size can be much larger.

   $$n > b > b_0$$

   where b_0 is the vector length that can fully utilize the data loading and processing pipeline. SR2201 is a pseudo–vector based MPP machine since it has 128 physical registers with pseudo–vector function, thus the larger the local block size, the higher the local computation speed.

2. Then, it is necessary to determine the message passing saturation message length L_0 and the relation function between block size b, problem dimension n, processor grid $r \times c$ and message length L.

   $$L = f_1(n, r, c, b)$$

   After this, we set $L > L_0$, then the proper block size that maximizes the utilization of communication network can be determined. Thus we have

   $$b > f_1^*(n, r, c, L_0)$$

3. Thirdly, the memory requirement constraint is added to block size b. Thus we have:

   $$M(n, p, r, c, b) < m$$

   This requires the b satisfies the following:

   $$b < M^*(n, p, r, c, m)$$

4. Then the load balance constraint must be considered since our goal is the minimal parallel execution time. For the two dimensional block cyclic data distribution, the b must be smaller enough such that there are enough number of blocks to be distributed cyclically for several times k. The k is one balance parameter need to be determined for specific application, processor grid, computation speed and communication speed. We can find out the changing rules of k through experiments on small scale problem size and small scale processor number. Thus we have

   $$f_2(r, c)kb < n$$

   i.e.

   $$b < \frac{n}{f_2(r, c)k}$$

5. Finally, summarizing all these constraints, we have the following constraints for block size b:

   $$\begin{cases} b > max(l_b, f_1^*(n, p, L_0)) \\ b < min(M^*(n, p, r, c, m), \frac{n}{f_2(r,c)k}, u_b, n) \end{cases}$$

If there are solutions for these inequalities, then the range of optimal block size can be found. The user can select the suitable one from the values lies in the range. However, when there are no solution that satisfy all constraints, the users has to determine the most important constraints among them through matching experiments results of small scale problem size and system configuration with these constraints. Then the optimal or sub-optimal block size that can satisfy the dominant constraints can be selected while still keeping in mind the requirement of other constraints.

4. CASE STUDY

In this section, we use the double precision(D) timing drivers parallel LU and parallel QR(DLU and DQR for short) in ScaLAPACK as examples to illustrate the framework application process. The detailed algorithm description of parallel blocked LU and QR factorization can be found in [3, 5, 6, 15]. The MPI BLACS package [17] will be used as the message passing platform. We first introduce the experimental platforms used in our experiments, then introduce how to identify the required framework parameters step by step on SR2201. Finally, we would give the prediction formulas of optimal parallel block size for DLU and DQR on SR2201 and DAWNING 3000 respectively, then the prediction results will be compared with the measured results.

4.1 Experimental Platforms

4.1.1 HITACHI SR2201

One experimental machine we used is a pseudo-vector processing based MPP machine – HITACHI SR2201. It is the second generation MPP machine of HITACHI with 3D–Crossbar interconnection network and can be scaled up to 2048 processors with peak performance 614GFlops , each is one HARP 1E processor with clock speed at 150MHz, with 256MB main memory, 16KB/16KB L1 cache(Data/Instruction split) with cache line size 128Bytes, and 512KB off-chip L2 cache, peak performance 300MFlops and augmented with pseudo–vector processing capability [7]. The one direction communication bandwidth of SR2201 is 300MB/s. And it supports Remote DMA, PVM and MPI message passing environments. The system we used has 32 processors, 30 can be used for parallel computing and 2 for system and I/O services.

4.1.2 DAWNING3000

Another experimental machine we used is a RISC-based SMP Cluster platform – DAWNING 3000 SuperServer. The full scale system has 64 four-way SMP computing nodes, each processor is 375MHZ Power3-II with 64KB/32KB L1 Data/Instruction split cache and 4MB L2 cache; each node has 2GB main memory and 9GB SCSI Disk. Its peak performance is 403.2GFlops. The operating system is IBM AIX4.3.3. This system has compilers for C++, C, Fortran, and Java. Its interconnection network is Myrinet, the one direction communication bandwidth of DAWNING 3000 is 1.28Gbps. It supports parallel programming environments such as PVM, MPI and mathematical libraries such as BLAS, ESSL, ScaLAPACK, GAUSS98, etc.. The system we used has 4 nodes(16 processors).

In the following sections, we will demonstrate how to apply our framework to DLU and DQR on SR2201 and DAWNING3000.

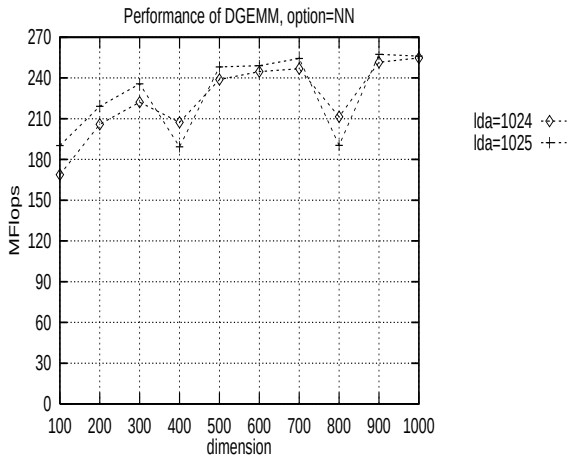

Figure 1: Performance Versus Block Size of `DGEMM(NN)` on SR2201

4.2 Local Block Size b_0

We use the BLAS3 subroutine `DGEMM`(Double Precision General Matrix Multiplication) to identify the optimal local computation block size range since it is frequently called by other subroutines in ScaLAPACK and dominant the computation cost. For DLU and DQR, this is especially true. Figure 1 depicts the performance of `DGEMM` with different block size using option `NN`(`N` for no transpose and `T` for transpose). The block size changed from 100 to 1000 step 100. The BLAS3 used in these experiments was the speedup version that our center(RDCPS) specifically developed for HITACHI SR2201 with "hill climbing" loop unrolling techniques [4, 11], which can utilize the pseudo–vector processing facility of SR2201 even better. The `lda` in these figures means the array leading dimension size, it is the column length of Fortran array. The `lda= 1024` will cause bank conflicts on SR2201 since it has 16 memory banks. From Figure 1, it can be easily seen that the performance increase with the increase of block size(problem size). This is due to the pipeline requirement of loop unrolling. However, the performance difference becomes little when block size larger than 300. This tells us that block size larger than 300 is enough to fulfill the pipeline of `DGEMM` on SR2201. Thus we have:

$$n \geq b_0 \geq 300$$

But this is not necessary condition since we can easily conclude that block size around 100 also has acceptable performance. The block size constraint can be tuned down to 100 sometimes necessary.

4.3 Message Passing Saturation Message Length L_0

The communication saturation message length L_0 of MPI was measured on SR2201 using round–trip ping–pong testing. This testing reflects the basic message passing performance of MPI on SR2201. The result is depicted in Figure 2.

From Figure 2, we can easily conclude that the saturation point L_0 for MPI is approximately around 1MBytes.

The relationship between message length L, processor grid shape $r \times c$, problem dimension n and block size b in DLU

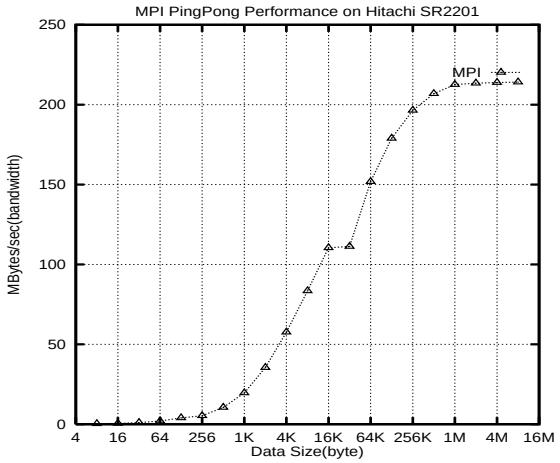

Figure 2: The Point–to–Point Performance of MPI on SR2201

and DQR of ScaLAPACK can be:

$$L_0 \leq L = \frac{enb}{max(r,c)}$$

thus we have

$$n \geq b_{mpi} \geq \frac{max(r,c)L_0}{en}$$

4.4 Memory Requirement

The memory requirement of DLU and DQR can be expressed as follows:

$$M(n,p,r,c,b) = e(\frac{n^2}{p} + \frac{nb}{min(r,c)} + \alpha) \leq m$$

where α is the dynamic memory used in communication buffer and other undetermined memory allocation and its value is relatively small. The m on SR2201 which represents user allocable maximize memory without swapping is around 150MB according to our experience, and it is 80MB on DAWNING3000. Thus we have:

$$b_{mem} \leq min(n, \frac{(\frac{m}{e} - \frac{n^2}{p} - \alpha) * min(r,c)}{n})$$

4.5 Load Balance

4.5.1 DLU

Considering the load balance of LU in ScaLAPACK, we need to determine the smallest k that can keep better load balance of LU. The factor k refers to the times that the block cyclic data distribution wraps around the column of processor grid. Thus the block size must satisfy

$$b_{bal} \leq \lceil \frac{n}{ck} \rceil$$

where b_{bal} is the largest possible block size that can still keep load balance, n is the array dimension size which is distributed, c is the number of processors in a row of $r \times c$ processor grid.

There are several papers [3, 6, 14] that analyzed the computational complexity of parallel blocked LU factorization using block cyclic data distribution, and the parallel scheme of block LU factorization can be separated into three major stages, they are:

- LU factorization on block column matrix with particpating of one column of processes(r), its total complexity is about $\frac{n^2 b}{2}$;
- Multiple right hands solving of triangular system with participating of one row of processes(c), its total complexity is about $\frac{n^2 b}{2}$;
- Updating of the remaining lower right sub-matrix with participating of all parallel processes(p), its total complexity is about $\frac{2}{3}n^3 - n^2 b$;

The load imbalance in parallel block LU is largely caused by the first two stages since they usually have slow local computation speed(non–block and bad memory access pattern algorithm), involving frequent fine grained communication(partial pivoting) and less parallelism(only one row or column of processes involved in). The larger the block size, the larger part of the total computation complexity will use slow computation and less parallelism. This requires the third stage must be the dominant part of the total parallel computation complexity, thus the largest b_{bal} must satisfy

$$\frac{\frac{2}{3}n^3 - n^2 b_{bal}}{p} >> \frac{n^2 b_{bal}}{2r} + \frac{n^2 b_{bal}}{2c}$$

Solving this constraint, we have:

$$\theta = \frac{3(r+c+2)b_{bal}}{4n} << 1$$

since

$$b_{bal} \leq \lceil \frac{n}{ck} \rceil$$

then we have

$$\theta = \frac{3(r+c+2)}{4kc} << 1$$

The θ is the parameter that depends on specific parallel computing platform, and it indicates the tolerable complexity ratio between the first two stages and the third stages, which we believe can be determined through experiments. If we can identify the largest value, θ_{max}, of θ, then the smallest k is

$$k = \frac{3(r+c+2)}{4\theta_{max} c}$$

Using this k, we can easily find the constraint b_{bal} as

$$b_{bal} \leq \lceil \frac{n}{ck} \rceil = \lceil \frac{4n\theta_{max}}{3(r+c+2)} \rceil$$

Since the larger the block size on SR2201, the higher the local computation speed, the optimal block size would be b_{bal} if the load balance constraint is the dominant factors in the selection of optimal block size.

4.5.2 DQR

The description of parallel QR factorization using block cyclic distribution was given in [5]. The algorithm can be roughly seperated into three stages:

1. QR factorization on one block column matrix participated by one column processes(r), its total complexity is $n^2b - \frac{2}{3}nb^2 + \frac{n^2}{2b} + nb$;

2. The computation of matrix T by one column of processes which owned the Householder vector V [15], its total complexity is $\frac{(n-b)^2b}{2}$;

$$Q = (I - \tau_1 v_1 v_1^T)(I - \tau_2 v_2 v_2^T) \cdots (I - \tau_{b-1} v_{b-1} v_{b-1}^T) = I - VTV^T$$

and

$$P = I - \tau v v^T$$

then

$$Q_+ = QP = I + V_+ T_+ V_+$$

among that $V_+ = [V \quad v]$,

$$T_+ = \begin{bmatrix} T & z \\ 0 & \rho \end{bmatrix}$$

and $\rho = -\tau$, $z = -\tau T V^T v$.

3. Q^T modification on the lower right corner of matrix A participated by all processes, its total complexity is $\frac{4}{3}(n-b)^3 + (n-b)^2 b$;

   - The process which owns the vector V broadcast in its process column, all processes compute $W = V^T A_2$ in parallel. The partial production is summed by the current row process, its complexity is $\frac{2}{3}(n-b)^3$;
   - The process which owns T broadcast T in its process row and compute $W = T^T W$, its complexity is $(n-b)^2 b$;(one row processes)
   - The processes in current process row broadcast W column-wise and all processes perform the local modification on $A_2 (A_2 = A_2 - VW)$, its complexity is $\frac{2}{3}(n-b)^3$;

Similar with DLU, the b_{bal} of DQR must satisfy the following conditions:

$$\frac{\frac{4}{3}(n-b_{bal})^3}{p} >> \frac{\frac{1}{2}(n-b_{bal})^2 b_{bal} + n^2 b_{bal}}{r} + \frac{(n-b_{bal})^2 b_{bal}}{c}$$

Since $b_{bal} << n$, we have:

$$\frac{\frac{4}{3}n^3}{p} >> \frac{3n^2 b_{bal}}{2r} + \frac{n^2 b_{bal}}{c}$$

Solving this constraint, we have

$$\theta = \frac{3(3c+2r)b_{bal}}{8n} << 1$$

since

$$b_{bal} \leq \lceil \frac{n}{ck} \rceil$$

then we have

$$\theta \leq \frac{3(3c+2r)}{8kc} << 1$$

If we can identify the largest value, θ_{max}, of θ, then the smallest k is

$$k = \frac{3(3c+2r)}{8\theta_{max} c}$$

Using this k, we can easily find the constraint b_{bal} as

$$b_{bal} \leq \lceil \frac{n}{ck} \rceil = \lceil \frac{8n\theta_{max}}{3(3c+2r)} \rceil$$

4.6 All Constraints

4.6.1 *DLU*

Adding all these constraints together, we plotted them in one figure for small processor grid 1×4 and 2×2. These graphs are given in Figure 3.

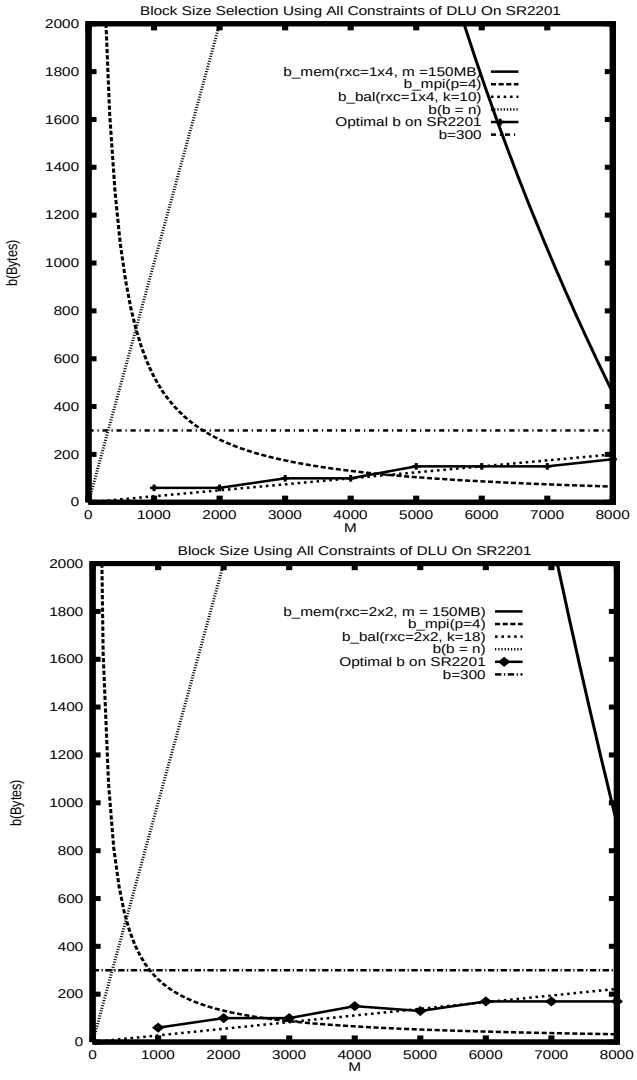

Figure 3: Block Size Selection of DLU under All Constraints on SR2201(r×c=1×4 and r×c=2×2)

Table 2: k and θ_{max} Values of DLU for Different Processor Grids on SR2201

Items	p_r	p_c	k	θ_{max}
1	1	4	10(10.5)	0.131(0.125)
2	2	2	18	0.125
3	1	8	8(8.25)	0.129(0.125)
4	2	4	12	0.125

From Figure 3, we can see that there is no large intersection area for these all constraints with load balance constraints, this fact tells us that there is no common block size area that can satisfy all these constraints at the same time. We have to trade off among all these constraints to select block size that satisfy the most important constraints.

The experimental measured optimal block sizes are also plotted in these figures to indicate the optimal block size with solid lines. To match the load balance line with these solid lines, we properly selected the k parameter for each case. An obvious trend in suitable k selection is that it will decrease with the increasing of the number of processor column and increase with the increasing of the number of processor row. This can be easily concluded from the above formula of k.

However, one obvious phenomenon from these figures is that the optimal block size is largely constrained by the load balance constraints, sometimes even along the load balance line when there are small intersection area among these constraints. This seems tell us that the load balance constraint is the dominant constraint for optimal parallel block size selection problem on SR2201. This stimulates us to find out the θ_{max} value on SR2201 using the k value that matches optimal block size well in experiment, and to see whether it is a constant or with little variant with different processor grid shape and problem size.

The k values for different processor grids and the corresponding θ_{max} values are listed in Table 2.

We can observe from Table 2 that the θ_{max} value on SR2201 would be around 0.125 after little tuning on k value. Thus we can set θ_{max} to be $\frac{1}{8}$ and get the following b_{bal} formula

$$b_{bal-sr2201_{lu}} \leq \frac{n}{6(r+c+2)}$$

As we stated above, the largest b_{bal} would be the optimal block size of parallel LU on SR2201 since the load balance constraint matches the experimental optimal block size well. But we need to add two additional conditions on SR2201 to obtain the optimal block size $b_{opt-sr2201_{lu}}$:

1. Weak condition:

$$mod(n, b_{opt-sr2201_{lu}}) = 0$$

2. Strong Condition:

$$mod(b_{opt-sr2201_{lu}}, 16) \neq 0$$

To compare the predicted and experimental optimal block size, we listed them in Table 4 for large scale system configuration and problem size. In the following tables, without specific statement, the figures in bracket is the measured performance(MFlops) at the optimal block size(Experimental line) and the percent of performance difference between the predicted and measured optimal block size(Predicted line). `NA` represents Not Available. For cases that predicted block size matched perfectly with optimal block size, we only gave the block size.

Table 3: k and θ_{max} Values of DQR for Different Processor Grids on SR2201

Items	r	c	k	θ_{max}
1	1	4	13	0.098
2	2	2	24	0.079
3	4	1	50	0.0833
4	1	8	12	0.1026
5	2	4	18	0.084
6	4	2	31	0.086

Using the experiences obtained from SR2201, we only need to get the value of θ_{max} after careful tuning on k value of DAWNING3000. Though on RISC-based MPP machines, it seems more difficult than on PVP-based MPP to obtain a constant θ_{max}, since the optimal block size must be the power of 2, which gives little space to tune the proper k value. Thus by choosing commonly acceptable θ_{max} to be 0.06, we could get the following $b_{bal-dawn3000_{lu}}$ formula

$$b_{bal-dawn3000_{lu}} \leq \frac{2n}{25(r+c+2)}$$

As we stated above, the largest $b_{bal-dawn3000_{lu}}$ would be the optimal block size of parallel LU on DAWNING3000 since the load balance constraint matches the experimental optimal block size well. But we need to add two additional conditions on DAWNING3000 to obtain the optimal block size $b_{opt-dawn3000_{lu}}$:

1. Weak condition:

$$mod(n, b_{opt-dawn3000_{lu}}) = 0$$

2. Strong Condition:

$$2^{n_1-1} < b_{bal-dawn3000_{lu}} \leq b_{opt-dawn3000_{lu}} = 2^{n_1}$$

n_1 is integer.

Where the first condition on b_{opt} is only for even data distribution and can be ignored if the performance acceptable, while the second condition must be obeyed on DAWNING3000 to obtain better cache performance. These two additional conditions on SR2201 and DAWNING3000 can be applied to DQR as well and we will not give these additional conditions again in the next section.

To compare the predicted and experimental optimal block size, we listed them in Table 5 for large scale system configuration and problem size.

4.6.2 *DQR*

The k values for different processor grids and the corresponding θ_{max} values on SR2201 are listed in Table 3.

We can observe from Table 3 that the average θ_{max} value on SR2201 would be around 0.089 after little tuning on k value. Thus we can set θ_{max} to be 0.09 and get the following b_{bal} formula

$$b_{bal-sr2201_{QR}} \leq \frac{6n}{25(2r+3c)}$$

To compare the predicted and experimental optimal block size, we listed them in Table 6 for large scale system configuration and problem size.

Using the experiences obtained from SR2201, we get the commonly acceptable value of θ_{max} after careful tuning on k value on DAWNING3000. Though it is more difficult on DAWNING3000 than on SR2201 to obtain a nearly constant θ_{max}. By choosing the θ_{max} to be 0.06, we could get the following $b_{bal-dawn3000_{qr}}$ formula

$$b_{bal-dawn3000_{qr}} \leq \frac{4n}{25(2r+3c)}$$

To compare the predicted and experimental optimal block size, we listed them in Table 7 for large scale system configuration and problem size.

From Table 4 to Table 7, we can easily find out that most of the predicted optimal block size matched well with measured optimal block size, except with little performance differences for unmatched case (mostly far less than 10.0%). Some can even perfectly matched. This demonstrates the accuracy and usability of our framework in finding the formula of optimal block size on SR2201 and DAWNING3000. On SR2201, for DLU, the average percent of performance differences between predicted and experimental is 1.71%; For DQR, the average is 2.41%. On DAWNING3000, for DLU, the average percent of performance differences between predicted and experimental is 3.68%; For DQR, the average is 1.63%. However, for small problem size using larger number of processors, especially for those using long flat processor grid, the prediction deviates a lot from measured performance(not given since space limitation). This is largely due to the too small local block size selection of our framework in order to keep better load balance under such extreme condition and the sharply local computational speed decreasing at too small block size. This tells us that the local computational speed will become the dominate factor under some extreme conditions.

Above all, we got the following principles in selecting optimal block size for ScaLAPACK based applications:

- For applications using large amount of processors, to split work evenly among processors, i.e., to keep load balance would be the most important constraint among all factors that affect the selection of optimal block size. The load balance line can be used as the indication of possible optimal block size area; The optimal block size won't be larger than the allowed block size that can keep load balance.
- The selection of k in load balance constraint is a key factor that affects the accuracy of optimal block size selection. It depends on the specific application, machines and interconnection network features on which the application will run; The experience of analyzer is also important in identifying it;
- Memory constraint is not so important among these cases we studied since the problem size is relative small. It will become important when the problem size becomes larger;
- The message passing saturation length constraint is not so important factor for dense matrix based application since the dominant part of application execution time would be local computation time. It will become important when the ratio of communication to computation becomes larger.
- The lower bound on local optimal computation block size is not the lower bound on parallel optimal block size selection as we would expect. The load balance constraint limits its usage. However, it prevents load balance constraint from selecting too small block size to keep better load balance.
- On SR2201 and DAWNING3000, the optimal parallel block size selection is mainly influenced by processor grid shape and problem size.

5. CONCLUSIONS

Optimal block size selection is a complicate problem in single processor as it will change with different application loop characteristics, machine architecture and memory hierarchy. It is still a difficult task in parallel computing since it becomes the interaction between local block size problem, load balance, memory requirement and communication systems. In this paper, we try to address this problem for parallel LU and QR factorization based on ScaLAPACK package since it uses block cyclic data distribution fashion, block size plays important role in determining the final application performance. Through analysis and experiments, we find out that among all these factors, load balance constraint plays key role in determining the optimal parallel block size selection. After matching the measured optimal block size on small scale processor configuration, we finally found out the rules on the selection of θ_{max} parameter. Simple optimal block size prediction formula were given for parallel DLU and DQR on SR2201 and DAWNING3000 respectively, whose prediction matched well with experimental results.

The local computation optimal block size can not be easily satisfied because of load balance requirement, but it still prevent load balance constraint from selecting too small block size. This tells the truth that load balance is more important than local faster computation speed on SR2201 and DAWNING3000, the relative not so larger performance difference among local block size changes plays another role in these results.

The memory constraint and communication message length constraint play ignorable roles in our analysis since they will become important for application with larger problem size and larger communication to computation ratio.

Further research on other timing drivers in ScaLAPACK package and the application of our framework on other parallel machines would be our future works.

6. REFERENCES

[1] L. Blackford, J.Choi, A.Cleary, J.Demmel, I.Dhillon, and etc. Scalapack: A portable linear algebra library for distributed memory computers - design issues and performance. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*, pages 5–5. ACM SIGARCH and IEEE Computer Society, November 1996.

[2] S. Carr and K. Kennedy. Improving the ratio of memory operations to floating point operations in loops. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(6):1768–1810, November 1994.

[3] X. Chi. Parallel implementation of *lu* factorization. In *Proceeding of the Second IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 108–112. IASTED, December 1998.

[4] X. Chi, Y. Li, J. Sun, Y. Zhang, and P. Zhu. Developing high performance blas and scalapack on hitachi mpp sr2201. In *Proceeding of 3rd High Performance Computing Asia Conference and Exhibition*, pages 142–151. National Supercomputing Research Centre, Singapore, September 1998.

[5] J. Choi, J. J. Dongarra, and etc. The design and implementation of the scalapack *lu*, *qr*, and cholesky factorization routines. *Scientific Programming*, 5(3):173–184, September 1996.

[6] F. Desprez, S. Domas, and B. Tourabcheau. Optimal data distribution for lu decomposition routine using communication/computation overlap. In *Research Report No. 3094*, pages 1–20. National Institute of Information and Automation, France, February 1997.

[7] H. Fujii, Y. Yasuda, and etc. Architecture and performance of the hitachi sr2201 massively parallel processor system. In *Proceeding of 11th International Parallel Processing Symposium (IPPS '97)*, pages 233–241. IEEE Computer Society, April 1997.

[8] K. Gallivan, W.Jalby, U.Meier, and A. Sameh. Impact of hierarchical memory systems on linear algebra algorithm design. *The International Journal of Supercomputer Applications*, 2(1):12–48, March 1988.

[9] D. Gannon, W. Jalby, and K. Gallivan. Strategies for cache and local memory management by global program transformation. *Journal of Parallel and Distributed Computing*, 5(5):587–616, October 1988.

[10] M. S. Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. In *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 63–74. ACM Press, April 1991.

[11] Y. Li and P. Zhu. Speedup methods and implementation techinques of blas. *Chinese Journal of Numerical Methods and Computer Applications*, 19(3):227–240, September 1998.

[12] W. Lichtenstein and S. L. Johnsson. Block-cyclic dense linear algebra. *SIAM Journal on Scientific Computing*, 14(6):1259–1288, November 1993.

[13] N.Mitchell, K.Hogstedt, L.Carter, and J.Ferrante. Quantifying the multi-level nature of tiling interactions. *International Journal of Parallel Programming*, 26(6):641–670, December 1998.

[14] T. Rauber and G. Runger. Optimal data distribution for *lu* decomposition. In *Proceeding of the EuroPar'95 Conference, Lecture Notes in Computer Science NO.966*, pages 391–402. Springer, August 1995.

[15] R. Schreiber and C. V. Loan. A storage efficient wy representation for products of householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, January 1989.

[16] P. Strazdins. Matrix factorization using distributed panels on the fujitsu ap1000. In *Proceeding of the IEEE First International Conference on Algorithms and Architectures for Parallel Processing*, pages 263–273. IEEE Computer Society, April 1995.

[17] R. C. Whaley. Basic linear algebra communication subprograms: analysis and implementation across multiple parallel architectures. In *LAPACK Working Note 73, Technical Report: UT-CS-94-234*. University of Tennessee, Knoxville, TN, USA, May 1994.

[18] M. Wolf and M. Lam. A data locality optimization algorithm. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 30–44. ACM SIGPLAN, June 1991.

[19] M. J. Wolfe. Iteration space tiling for memory hierarchies. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 357–361. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, December 1987.

[20] M. J. Wolfe. More iteration space tiling. In *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 655–664. ACM/IEEE, November 1989.

Table 4: Predicted and Experimental Optimal Block Size of Parallel DLU on SR2201

$r \times c$ N	6000	7000	8000	9000	10000
1×8(P)	100	100	120(-0.8%)	140	152
(E)	100	100	100(1173)	NA	NA
2×4(P)	125(-7.7%)	140	170	190	200
(E)	140(1142)	140	170	NA	NA
1×16(P)	50(-2.5%)	60	70	90	90(-0.1%)
(E)	60(1708)	60	70	90	100(2063)
2×8(P)	84	100	110	125(-5.2%)	139(-8.7%)
(E)	84	100	110	120(2204)	140(2306)
4×4(P)	100	117(-5.4%)	134(-0.9%)	150	167(-6.8%)
(E)	100	110(2046)	150(2195)	150	180(2322)
4×6(P)	84(-1.3%)	100	110(-0.8%)	125(-6.8%)	140(-0.3%)
(E)	100(2647)	100	100(3027)	130(3143)	130(3265)
3×10(P)	60(-3.1%)	70	90(-2.1%)	100	100(-0.6%)
(E)	84(3048)	70	70(3471)	100	90(3862)

Table 5: Predicted and Experimental Optimal Block Size of Parallel DLU on DAWNING3000

	6000	7000	8000	9000	10000
1×8(P)	64	64	64	128(-14.8%)	128
(E)	64	64	64	64(5949)	NA
2×4(P)	64	128(-6.4%)	128(-4.6%)	128(-4.5%)	128
(E)	64	64(5627)	32(5668)	64(5919)	NA
4×2(P)	64(-4.0%)	128(-5.6%)	128(-10.4%)	128(-9.9%)	128
(E)	32(4330)	64(4528)	32(4641)	64(5305)	NA
8×1(P)	64(-10.1%)	64(-5.3%)	64(-5.5%)	128(-6.3%)	128
(E)	16(3229)	32(3416)	32(3281)	32(3495)	NA
1×16(P)	32	32(-2.14%)	64	64	64
(E)	32	64(8885)	64	64	64
2×8(P)	64	64	64	64	128(-8.98%)
(E)	64	64	64	64	64(10509)
4×4(P)	64(-8.78%)	64	64(-1.25%)	128(-10.2%)	128(-7.9%)
(E)	32(6516)	64	32(7749)	64(8838)	32(8632)
8×2(P)	64(-2.4%)	64	64(-9.2%)	64	128
(E)	32(5221)	64	32(5871)	64	NA
16×1(P)	32	32	64(-4.1%)	64(-3.0%)	64(-2.89%)
(E)	32	32	32(4051)	32(4885)	32(4748)

Table 6: Predicted and Experimental Optimal Block Size of Parallel DQR on SR2201

	6000	7000	8000	9000	10000
1×8(P)	60(-1.49%)	70	70(-1.89%)	84(-0.37%)	90(-1.94%)
(E)	70(1205)	70	90(1268)	90(1344)	84(1341)
2×4(P)	90	100(-0.07%))	120(-2.04%)	130(-1.03%)	150
(E)	90	90(1399)	90(1422)	90(1460)	150
1×16(P)	30(-5.75%)	30(-8.95%)	40(-4.52%)	40(-4.33%)	50(-2.69%)
(E)	60(2019)	60(2178)	50(2166)	60(2331)	60(2340)
2×8(P)	50(-2.02%)	60(-3.98%)	70	84(-3.37%)	84(-2.48%)
(E)	70(2371)	70(2486)	70	90(2638)	70(2666)
8×2(P)	60(-2.13%)	76(-4.28%)	90	100(-0.64%)	110(-2.76%)
(E)	84(2399)	84(2500)	90	90(2666)	90(2750)
16×1(P)	40(-11.16%)	50(-4.16%)	50(-3.97%)	60(-2.33%)	68
(E)	60(2034)	70(2163)	90(2265)	90(2356)	NA
4×4(P)	70(-1.56%)	84	100(-8.30%)	100(-0.69%)	120(-2.3%)
(E)	100(2495)	84	90(2710)	90(2772)	100(2820)
4×6(P)	60	70	70	83(-3.21%))	90
(E)	60	70	70	90(3923)	90
3×10(P)	40(-4.73%)	50(-1.95%)	50(-1.79%)	60(-0.72%)	70(-2.64%)
(E)	60(4035)	70(4262)	70(4414)	70(4577)	90(4692)

Table 7: Predicted and Experimental Optimal Block Size of Parallel DQR on DAWNING3000

	6000	7000	8000	9000	10000
1×8(P)	64(-3.33%)	64(-1.4%)	64(-1.8%)	64	64
(E)	32(6930)	32(7075)	32(7077)	NA	NA
2×4(P)	64	128(-8.2%)	128(-3.4%)	128(-1.2%)	128
(E)	64	64(7468)	64(7570)	64(7736)	NA
4×2(P)	128(-2.7%)	128(-1.3%)	128(-4.1%)	128(-3.6%)	128
(E)	64(6794)	64(7035)	64(7253)	64(7389)	NA
8×1(P)	64	64	128(-3.2%)	128(-1.5%)	128
(E)	64	64	64(6325)	64(6523)	NA
1×16(P)	32	32(-8.6%)	32	32	32
(E)	32	64(10255)	32	32	32
2×8(P)	64	64	64	64	64
(E)	64	64	64	64	64
4×4(P)	64	64	64	128(-2.99%)	128(-3.0%)
(E)	64	64	64	64(12813)	64(13175)
8×2(P)	64	64	64	128(-3.4%)	128(-6.1%)
(E)	64	64	64	64(11725)	64(12280)
16×1(P)	32(-1.4%)	32(-1.8%)	64	64(-0.1%)	64(-0.9%)
(E)	64(7910)	64(8557)	64	32(9226)	32(9649)