Approximations for the General Block Distribution of a Matrix

Bengt Aspvall* Magnús M. Halldórsson†* Fredrik Manne*

August 12, 1998

Abstract

The general block distribution of a matrix is a rectilinear partition of the matrix into orthogonal blocks such that the maximum sum of the elements within a single block is minimized. This corresponds to partitioning the matrix onto parallel processors so as to minimize processor load while maintaining regular communication patterns. Applications of the problem include various parallel sparse matrix computations, compilers for high-performance languages, particle in cell computations, video and image compression, and simulations associated with a communication network.

We analyze the performance guarantee of a natural and practical heuristic based on iterative refinement, which has previously been shown to give good empirical results. When p^2 is the number of blocks, we show that the tight performance ratio is $\theta(\sqrt{p})$. When the matrix has rows of large cost, the details of the objective function of the algorithm are shown to be important, since a naive implementation can lead to a $\Omega(p)$ performance ratio. Extensions to more general cost functions, higher-dimensional arrays, and randomized initial configurations are also considered.

In comparison, Khanna et al. have shown the problem to be approximable within $O(1)$ factor [7], while Grigni and Manne have shown it to be NP-hard to approximate within any constant factor less than two [4].

1 Introduction

A fundamental task in parallel computing is the partitioning and subsequent distribution of data among processors. The problem one faces in this operation is how to balance two often contradictory aims: finding an equal distribution of the computational work and at the same time minimizing the imposed communication. In a data parallel computing environment, the running time is dominated by the processor with the maximal load, thus one seeks a distribution where the maximum load is minimized. On the other hand, blindly optimizing this factor, may lead to worse results if communication patterns are ignored.

We assume we are given data in the form of a matrix, with communication only involving adjacent items. This is typical for a large class of scientific computational problems. The partition that minimizes communication load is the *uniform partition*, or *simple block distribution*, where the n by n matrix is tiled by n/p by n/p squares. For instance, this improves on the *one-dimensional* partition, where n/p^2 columns are grouped together. However, workload, which we typically measure as the number of non-zero entries in a block, may be arbitrarily unbalanced, as non-zero entries may be highly clustered.

A partition that yields greatly improved workload is the *general block distribution*, where the blocks are arranged in an orthogonal, but unevenly spaced, grid. It can be viewed as an ordinary

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway.
{Bengt.Aspvall, Fredrik.Manne}@ii.uib.no
†Science Institute, University of Iceland, IS-107 Reykjavik, Iceland. mmh@rhi.hi.is.

block partitioning of an array where one allows the dividers for one column (or row) block to be moved simultaneously. The advantage of this distribution is that it preserves both the locality of the matrix and the array-structured communication of the block distribution while at the same time allowing for different sized blocks.

If the underlying problem has a structure such that communication is local, using a rectilinear partitioning gives a simple and well structured communication pattern that fits especially well on grid connected computers. The simplicity of the general block distribution also makes it possible for compilers to schedule the communication efficiently. It has therefore been included as an approved extension for data mapping in High Performance Fortran HPF2 [5].

Applications of the general block distribution include various parallel sparse matrix computations, compilers for high-performance languages, particle in cell computations, video and image compression, and simulations associated with a communication network [1, 7, 8, 5, 11]. See [9] for a discussion of other rectilinear partitioning schemes.

Computing the optimal general block distribution was shown to be NP-hard by Grigni and Manne [4]. In fact, their proof shows that the problem is NP-hard to approximate within any factor less than 2. Khanna et al. [7] have shown the problem to be constant-factor approximable. They did not give a bound on the value of the constant attained by their algorithm, but an examination of their analysis appears to give a bound of 127. They also did not try to analyze the complexity of the algorithm, but it is specified in terms of a collection of submatrices that can be asymptotically $O(n^4)$ or square in the size of the input. They additionally indicated a simple $O(\log^2 n)$-approximate algorithm, also defined on a quadratic size collection.

The subject of the current paper is a performance analysis of a heuristic that has been considered repeatedly in the applied literature. This *iterative refinement* algorithm was given by Nicol [11], and independently by Mingozzi et. al. [10] and Manne and Sørevik [9]. It is based on iteratively improving a given solution by alternating between moving the horizontal and vertical dividers until a stationary solution is obtained. The algorithm and the general block distribution are given in Section 2.

We analyze the basic iterative refinement algorithm in Section 3.2 and find that it yields a performance ratio of $\theta(\sqrt{p})$ when the cost of each row is not a significant fraction of the whole instance. On the other hand, the performance deteriorates in instances with very heavy rows, and becomes as poor as $\theta(p)$.

In order to combat this weakness, we modify or constrain the operation of the algorithm in Section 3.3. We give two ways of modifying the objective functions of the one-dimensional subproblems that both lead to a $\theta(\sqrt{p})$ performance ratio on all instances.

We next consider the effect that starting configurations, or initial partitions, can have. In particular, a promising idea, suggested by Nicol [11], is to use random initial partitions, and possibly making multiple trials. We show in Section 3.4 that this is not beneficial, as the resulting performance ratio is $\Omega(p/\log p)$. Extensions of the results to more general cost functions and to matrices of higher dimensions are given in Section 3.5. The implementation of the algorithms is given in Section 4, with some improvements in the time complexity over previous work [11, 9].

Our analysis here indicates that the iterative refinement algorithm has a considerably weaker worst-case behavior than is possible by polynomial-time algorithms. Nevertheless, it may be valuable especially for small to moderate values of p, which is the case in our motivating application: load balancing on parallel computers. It is also quite efficient, being sub-linear except for a simple linear-time precomputation step. Finally, it is conceptually simple, natural enough to be discovered independently by at least three groups of researchers, easy to implement, and has been shown to give good results on various practical instances and test cases [11, 9].

2 The General Block Distribution

For integers a and b, let $[a, b]$ denote the interval $\{a, a+1, \ldots, b\}$.

For integers n and p, $1 \leq p \leq n$, a non-decreasing sequence $(1 = r_0, r_1, \ldots, r_p = n+1)$ of integers defines a *partition* of $[1, n]$ into the p intervals $[r_i, r_{i+1} - 1]$, for $0 \leq i < p$. For completeness, we allow empty intervals.

Definition 2.1 (General Block Distribution) *Given an n by n matrix A and integer p with $1 \leq p \leq n$, a* general block distribution *consists of a pair of partitions of $[1, n]$ into p intervals. It naturally partitions A into the p^2 contiguous blocks, or submatrices.*

A *block* is a submatrix outlined by pairs of adjacent horizontal and vertical dividers. A *column block* (*row block*) is a set of column between adjacent vertical (horizontal) dividers, respectively. A *row segment* is an intersection of a row and a block.

In a parallel environment the time spent on a computation is determined by the processor taking the longest time. The natural optimization problem is then to find a general block distribution that minimizes the maximum cost over all blocks. The cost is here taken to be the sum of of the elements in a block under the assumption that the entries of A are non-negative. The corresponding decision problem was shown in [4] to be NP-complete.

The *iterative refinement* algorithm consists of performing the following improvement step until none exists that further reduces the cost:

> With the vertical delimiters fixed, find an optimal distribution of the horizontal delimiters. Then, with the new horizontal delimiters fixed, do the same for the vertical delimiters.

Thus, the algorithm alternately performs vertical and horizontal *sweeps* until converging to a locally optimal solution. Each sweep can be viewed as a one-dimensional subproblem, for which efficient algorithms are known [2, 11, 12].

In the first vertical partition, A is partitioned optimally into p vertical intervals without the use of the horizontal delimiters. The number of iterations needed to obtain a converged solution varied between 2 and 13 in tests presented in [9].

For the remainder we may sometimes assume for convenience that we have p dividers (instead of $p-1$). Clearly this does not affect the asymptotic behavior. Note that the outlines of the matrix form additional dividers.

3 Performance Analysis

In this section, we analyze the performance guarantees of the iterative refinement algorithm and simple modifications thereof. We begin in Section 3.1 with intermediate results on the 1-D subproblem. We analyze in Section 3.2 the performance of the pure iterative refinement algorithm, which is dependent on the cost of the heaviest row of the input matrix. We then give in Section 3.3 simple modifications to the algorithm that yield better performance ratios when the input contains heavy rows. In Section 3.4 we consider strategies for the initial placement of vertical dividers, including uniform and random placement. We finally consider extensions of the problem in Section 3.5 to other cost functions and higher-dimensional arrays.

3.1 1-D Partitioning

As a tool for our studies of the general block distribution we need the following intermediate result on the one-dimensional case, i.e. how well a sequence of n non-negative numbers can be partitioned into p intervals. Let W be the sum of all the elements.

Lemma 3.1 *Given a positive integer p and a sequence of n non-negative numbers, a greedy algorithm yields a partition of the sequence into p intervals such that:*
(i) the cost of any interval excluding its last element is at most W/p, and
(ii) any interval with cost more than $2W/p$ consists of a single element.

Proof. Start the algorithm at the left end, and greedily add elements to form an interval until its cost exceeds $\frac{W}{p}$. If the cost is at most $2\frac{W}{p}$, make this the first interval, and inductively form $p-1$ intervals on the remaining array of total cost at most $W\frac{p-1}{p}$. If the cost exceeds $2\frac{W}{p}$, place a divider on both sides of the last element, forming two intervals. Then, inductively form $p-2$ intervals on the remaining array of cost at most $W\frac{p-2}{p}$. The only intervals that can have cost exceeding $2\frac{W}{p}$ are those formed by the last element added to a group, as in the second case. ∎

3.2 Pure iterative refinement

The performance ratio attained by the iterative refinement algorithm turns out to be highly dependent on the maximum cost of a row. If this cost is small, the performance is good, while it reduces to the trivial performance bound attained by the final 1-D sweep alone when the row cost is high.

Let us first notice that the cost of the optimal solution, OPT, is at least W/p^2, where W is the total cost of the matrix, since the number of blocks is p^2.

Theorem 3.2 *Let R denote the cost of the heaviest row, $R = \max_i \sum_j A[i,j]$. Then, the performance ratio of the pure iterative refinement algorithm equals $\theta(p)$, when $R = \theta(W/p)$, but only $\theta(\sqrt{p})$ when $R = O(W/p^{1.5})$.*

The theorem is established by the following three lemmas, along with the trivial $O(p)$ ratio obtained by a single 1-D sweep.

Lemma 3.3 *The performance ratio of pure iterative refinement is $\Omega(p)$ as a function of p alone.*

Proof. Consider the p^2 by p^2 matrix $A = (a_{i,j})$, where

$$a_{i,j} = \begin{cases} p^2, & \text{if } i = 1 \text{ and } j = (p-1)p+3, \ldots, p^2, \text{ or} \\ & j = 1 \text{ and } i = (p-1)p+3, \ldots, p^2 \\ 1, & \text{if } i, j = p+1, \ldots, p(p-1), \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

Observe that the cost of the first column and the cost of the first row are $(p-2)p^2 = W/p$. The iterative refinement algorithm will first assign the vertical dividers to be the multiples of p. One choice compatible with the definition of the algorithm for the assignment of the horizontal dividers assigns them also the multiples of p. The cost of this partition is the cost of the heavy row, or $W/p = (p-2)p^2$, and no improvements are possible that involve either horizontal or vertical dividers but not both.

The optimal partition consists of the vertical and horizontal dividers $p(p-1)+4, p(p-1)+6, \ldots, p^2-2$, followed by $3p, 5p, \ldots, (p-1)p$. The cost of this partition is $4p^2$, for an approximation ratio of $\frac{p-2}{4}$. For the upper bound, recall that a vertical (or horizontal) partition alone achieves a performance ratio $p+1$. ∎

When the cost of each row is bounded, the algorithm performs considerably better.

Lemma 3.4 *Following the first (vertical) sweep of the algorithm, there exists a placement of horizontal dividers such that the cost of blocks excluding the heaviest row segments within them is at most* $2(\sqrt{p}+1)OPT$.

Proof. We show the existence of a set of horizontal dividers that achieves the bound. The algorithm, which performs optimally under the given situation, will then perform no worse.

Let O denote the set of dividers, horizontal and vertical, in some optimal 2-D solution. We say that a column block is *thick*, if at least $\sqrt{p}$ vertical dividers from O go through it. Otherwise, a column block is *thin*. The solution we construct uses the even-numbered horizontal dividers from O, as well as $\sqrt{p}/2$ dividers for each of the thick column block to minimize the cost of its blocks.

Each block from a thin column block has at most one optimal horizontal divider and $\sqrt{p}-1$ vertical dividers from O crossing the block. Hence, the cost of the block is at most $2\sqrt{p}OPT$, or within the desired bound.

Each thick column block is of cost at most W/p plus a single column. The cost of each column segment is bounded by $2OPT$, given the even-numbered horizontal dividers from O. The cost of the rest of the block, excluding the cost of the heaviest row segment, is at most W/p divided by $\sqrt{p}/2$, or $2W/p^{1.5}$. Since $OPT \geq W/p^2$, this is at most $2\sqrt{p}OPT$. Thus, blocks in thick column blocks, excluding the heaviest row segment, are of cost at most $(2\sqrt{p}+2)OPT$. ∎

The lemma holds in particular for the iterative algorithm, thus we get good bounds when row cost is small.

Corollary 3.5 *When each row is of cost at most $O(W/p^{1.5})$ the iterative refinement algorithm achieves a performance ratio of $O(\sqrt{p})$ in two sweeps.*

For the case of small row cost, we get lower bounds that match within a constant factor.

Lemma 3.6 *The performance ratio of the iterative refinement algorithm is at least $\frac{\sqrt{p}}{4}$, even when the cost of each row is less than W/p^2.*

Proof. Consider the matrix A in Figure 1. We assume that p is a square number, and let $\alpha = \sqrt{p}$. The matrix A consists of a large block of dimension $\alpha(\alpha^2-\alpha+1) \times \alpha(\alpha^2-\alpha+1)$ in the upper left corner. The value of each element in this block is $1/(\alpha^2-\alpha+1)$. On the diagonal below the large block there are $\alpha-1$ blocks each of size $\alpha \times \alpha$. The value of each element in these blocks is 1.

The total cost W of A is $p^2 = \alpha^4$. The cost of each row is at most one, or W/p^2. That can be arbitrarily reduced further by making repeated copies of each column and row.

With these values the columns can be divided into $p = \alpha^2$ column blocks each consisting of α columns and of cost α^2. This is indicated by the dotted lines in Figure 1. Since each column interval has the same weight this is the initial partition that will be returned by the iterative refinement algorithm. When performing the horizontal partitioning the large block will now be regarded as having cost α^2. Thus from the horizontal point of view there are α blocks each of cost α^2. Dividing each small diagonal block into α intervals will give a cost of α for each

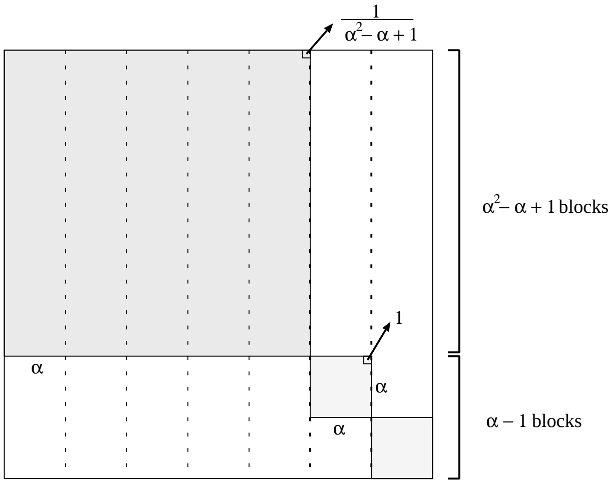

Figure 1: The array used for showing the lower bound.

block. Similarly using α intervals on the large block divides this into blocks of cost α. Note that it is possible to achieve this bound exactly since the number of rows in the large block is $\alpha(\alpha^2\alpha + 1)$. In this way we have used α^2 row blocks and achieved a solution where each block costs $\alpha = W/p^{1.5}$ giving a perfect load balance. Thus, this is the partition the algorithm will return after the first two sweeps. Returning to the vertical delimiters we cannot improve the solution further since each column block contains a block of cost α and the algorithm terminates.

Now, consider a solution where the large block is partitioned into blocks of size at most $2\alpha \times 2\alpha$. Then the cost of each block is at most $4\frac{\alpha^2-\alpha+1/4}{\alpha^2-\alpha+1} < 4$. Using $\frac{\alpha^2}{2}$ column and row blocks one is able to cover $\alpha^3 - \frac{\alpha^2}{2}$ rows/columns, which is less than the dimension of the large block. We now have at least $\frac{\alpha^2}{2}$ row and column blocks left to use on the $\alpha - 1$ small diagonal blocks. By using $\frac{\alpha}{2}$ horizontal and vertical delimiters on each of these we get 4×4 blocks of cost 4. Thus we see that there exists a solution of overall cost at most $4 = 4W/p^2$. ∎

This bound holds even when p is as large as $n/2$. We conjecture that $\sqrt{p}/4$ is indeed the tight performance ratio of the algorithm in general, up to a lower order term.

3.3 Modified iterative refinement algorithms

The lesson learned from Lemma 3.3 is that one may not blindly focus only on the heaviest column/row segment in each sweep; it is essential to balance also those segments that aren't immediately causing problems. In particular, although single heavy elements (or columns/rows) can cause the maximum block cost to be large, this should not be a *carte blanche* for the remaining partition to be arbitrarily out of balance.

We present two approaches for modifying the pure iterative refinement method, which both achieve a bound of $O(\sqrt{p})$. One makes a simple modification to the objective function, and yields the desired guarantee in three sweeps. The other requires only two sweeps to obtain an $O(\sqrt{p})$-approximate solution, but diverges slightly more from the original script.

A three sweep version We use a three sweep variant of the algorithm, where the first and the third sweep are as before, but the second sweep uses the following slightly modified objective function:

The cost of a block is the sum of all the elements in the block, excluding the heaviest row segment.

Lemma 3.7 *The above modified algorithm attains a performance ratio of* $4\sqrt{p}+4$.

Proof. By Lemma 3.4, the cost of any block after the second sweep is at most $(2\sqrt{p}+1)OPT$ plus the cost of a single row segment. We then only need to ensure that we reduce the cost of unusually heavy row segments in the third sweep, without affecting much the cost of the main parts of the blocks.

An assignment that contains every other of our previous vertical dividers, and every other of the vertical dividers from some optimal 2-D solution, ensures both: the cost of each block excluding the heaviest row segment at most doubles, while the cost of a row segment will be bounded by $2OPT$. Hence, the total cost of a block is at most $(2(2\sqrt{p}+1)+2)OPT \leq (4\sqrt{p}+4)OPT$. Since such an assignment exists for the third sweep, the optimal 1-D subroutine will find a solution whose cost is no worse. ∎

A two-sweep version We now consider an algorithm that works in two sweeps, as follows:

Step 1: Find the following two sets of vertical dividers independently:
(a) The $p/2$ dividers that minimize the maximum cost of any row segment.
(b) Use $p/2$ dividers that minimize the maximum cost of a column block.

Step 2: Find an optimal set of horizontal dividers.

We extend the analysis of the algorithm to its *performance function.* While the performance ratio of an algorithm is only a single value, describing the worst case ratio between the heuristic and the optimal values, the performance function $\rho(OPT)$ indicates the cost of the worst solution obtained by the algorithm for each possible optimal solution cost. In many cases, this yields a more informative analysis.

First, recall that $OPT \geq W/p^2$, and thus $\rho(OPT)$ is defined only for those values of OPT. Second, consider the case when $OPT \geq 2W/p$. There is an assignment of vertical dividers so that any column block of cost more than $2W/p$ will consists of a single column. A second sweep of horizontal dividers will then slice these separated columns optimally. Hence, $\rho(OPT) = 1$ when $OPT \geq 2W/p$.

We can generalize our analysis to show that $\rho(OPT) = O(\sqrt{\frac{W/p}{OPT}})$ for OPT in the range $[4W/p^2, 2W/p]$, providing a smoothly improving approximation bound.

Theorem 3.8 *The two-sweep algorithm has a performance function,*

$$\rho(OPT) = \max(O(\sqrt{\frac{W/p}{OPT}}), 1).$$

for each value of $OPT \geq 4W/p^2$.

Proof. As before, we present a particular set of horizontal dividers that achieve the bound, and thus claim that the algorithm performs no worse.

Part (a) of step one ensures that each row segment is of cost at most $2OPT$. Part (b) ensures that each column block is of cost at most $2W/p$ plus the cost of a single column (by Lemma 3.1).

Let $t = \sqrt{(W/p)/OPT}$. We now say that a column block is *thick*, if at least t of the optimal vertical dividers go through it, and otherwise *thin*. Observe that t is at most $\sqrt{p}$ since $OPT \geq W/p^2$.

We analyze the following set of horizontal dividers: Every other optimal horizontal divider, plus $\sqrt{p}/2$ dividers to minimize each of the at most $\sqrt{p}$ thick column blocks.

Using every other optimal horizontal dividers ensures that the cost of each column segment is at most $2OPT$, and that the cost of each thin block is at most $2tOPT$. Using t dividers to minimize the cost of blocks within each of the at most p/t thick column blocks ensures that those blocks are of cost at most $2/t$ times the cost of a column block, plus the cost of a column segment and the cost of a row segment. This is at most

$$\frac{2}{t} \cdot \frac{2W}{p} + 2OPT + 2OPT = (4t+4)OPT.$$

In particular, this is at most $(4\sqrt{p}+4)OPT$. ∎

This bound on the performance function can also be shown to be asymptotically tight.

3.4 Initial placement strategies

The iterative improvement method leaves open the possibility of using additional strategies for the initial placement of the vertical dividers. One approach would be to start with a *uniform placement*, with dividers at $n/p, 2n/p, \ldots, (p-1)n/p$. Nicol [11] suggests using *random placement*, where each divider is assigned a uniformly random value from 1 to n. He found this to give empirically good results. Random assignment also leaves open the possibility of repeating the whole improvement procedure, retaining the best of the resulting solutions.

Unfortunately, this approach does not improve the performance guarantee of the improvement method. In fact, with high probability, the performance ratio is decidedly worse, or $\Omega(p/\log p)$, which holds even if the procedure is repeated often. Basically, it suggests that any division strategy that is primarily based on the number of columns in each block is bound to fail. The strategy must rely on the weight of the columns. On the whole, however, we are led to the conclusion that partitioning methods that compute the horizontal and vertical dividers independently, cannot yield close to optimal approximations.

Theorem 3.9 *Random initial placement followed by iterative improvement has performance ratio $\Omega(p/\log p)$, expected and with high probability.*

Uniform initial placement followed by iterative improvement has performance ratio $\Theta(p)$.

The success of the algorithm on the example we shall construct depends on the size of the largest horizontal block in the initial partition. The following lemma bounds this value. Let ln denote the natural logarithm.

Lemma 3.10 *For a random partition of the sequence $1, 2, \ldots, n$ into p intervals, the probability that the largest interval contains at least $\alpha(n-1)\ln p/(p-1)+1$ numbers is at most $p^{-(\alpha-1)}$.*

Proof. Consider any fixed interval k, $1 \leq k \leq p$, and let X_k be the random variable denoting its length. We have

$$\mathrm{Prob}(X_k \geq d) = \binom{n-1-(d-1))}{p-1} \Big/ \binom{n-1}{p-1} = \prod_{i=0}^{p}(1 - \frac{d-1}{n-1-i}) \leq (1 - \frac{d-1}{n-1})^{p-1}.$$

Let $d_\alpha = \alpha(n-1)\ln p/(p-1)+1$, where $\alpha \geq 1$. Then

$$\text{Prob}(X_k \geq d_\alpha) \leq (1 - \frac{\alpha \ln p}{p-1})^{p-1} \leq e^{-\alpha \ln p} = p^{-\alpha},$$

using the fact that $1 - x \leq e^{-x}$. The probability that the largest of the p intervals is at least d_α is thus at most $p \times \text{Prob}(X_k \geq d_\alpha) \leq p^{-(\alpha-1)}$.

∎

Let E_n be the expected length of the largest of the p intervals for fixed p. The above lemma shows that $E_n \leq 2n \ln p/p$. A more precise bound is known:

$$\lim_{n\to\infty} \frac{E_n}{n} = \frac{H_p}{p},$$

where $H_p = \sum_{i=1}^{p} \frac{1}{i} = \ln p + O(1)$ is the p-th harmonic number. (Goulden and Richmond [3] posed this limiting identity as a problem in the American Mathematical Monthly with a solution submitted by Takács [13], and others. Holst [6] obtains the proof from a more general treatment of discrete spacings.) Thus, for fixed p the expected length of the largest interval is H_p times the length of an interval in the uniform partition.

We now prove the theorem.

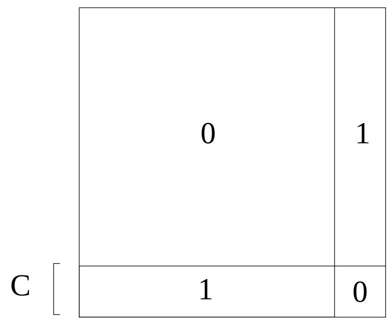

Figure 2: An example for which a uniform or random initial assignment leads to poor performance.

Proof. We assume that $p = o(\sqrt{n})$. Let $C = \sqrt{n}$.

Consider the $n \times n$ 0/1 matrix in Figure 2. Let us refer to the rightmost C columns as the *thick vertical block*, and the lowest C rows as the *thick horizontal block*. Only the elements in the symmetric difference between the two thick blocks have a cost one; the rest are zero elements. The cost of either block, denoted by Z, is thus $C \cdot (n - C) = n^{3/2} - n$.

Now consider the effect of a random assignment of vertical dividers. It is easy to deduce that with high probability no divider will hit the vertical heavy block. Let B denote the cost of the heaviest column block C_B. Let b satisfy $b = B(p-b)/Z$. We round b to the nearest integer.

After this first sweep, the algorithm proceeds deterministically. The second sweep must use b horizontal dividers on the thick horizontal block and $p-b$ on the thick vertical block. The cost of each block in the former is $B/b \approx Z/(p-b)$, while the cost of the latter is clearly $Z/(p-b)$.

On the third sweep, the algorithm must similarly use b vertical dividers on the thick vertical block and $p-b$ on the thick horizontal block. The cost of each block is then about $Z/b(p-b)$. No significant changes can then be made to either the horizontal or the vertical dividers independently to decrease this value.

We have skipped over the detail of the "joint block", the only block that contains elements from both thick blocks. Its size may bias the assignment of dividers somewhat, resulting in small oscillations. None of them can make significant difference, and in fact, cannot change the number of dividers used to partition either heavy block.

To wrap up this analysis, compare the algorithm's solution to the solution that on each side uses one divider to separate the heavy blocks and $p/2-1$ dividers on each of the them. The cost of this solution is then Z/p^2, and the ratio between the two solutions $p^2/(p-b)b \approx p/b$. From Lemma 3.10, with high probability, the value of b is $O(\ln p)$. Hence, the performance ratio is at least $\Omega(p/\ln p)$, expected and with high probability.

The proof of the lower bound for the uniform partition is left as an exercise. ∎

We can also observe that for any number of repetitions of this random procedure, within any polynomial of p, yields a performance ratio of at least $\Omega(p/\log p)$. We also remark that the ratio can be shown to be $\theta(p/\log p)$.

Remark: Recall that our basic iterative improvement algorithm starts with an optimal vertical partition without any horizontal dividers. We might view this as starting with a totally degenerate initial partition of the rows. On a random initial partition the algorithm initially performs more like on a uniform partition than when started with no horizontal dividers.

3.5 Extensions

Other cost functions While the sum of the elements within a block is usually the most natural measure, other cost functions may be more appropriate in certain applications. For some examples, see [12]. The results of this paper can easily be extended to other reasonable cost functions, in particular the following class.

Corollary 3.11 *General Block Partitioning can be approximated within $O(\sqrt{p})$ for any sub-additive cost function, i.e. if $\phi(B) \leq \phi(B_1) + \phi(B_2)$ whenever $B = B_1 \cup B_2$.*

Higher dimensions Our analysis of the 2-D case extends straightforward to higher dimensional matrices.

Claim 1 *The iterative refinement algorithm attains a performance ratio $\theta(p)$ on three-dimensional matrices.*

This generalizes to a ratio of $p^{d/2}$ for d-dimensional matrices. Matching lower bounds are straightforward generalizations of the 2-D case.

While this bound may appear weak, it is a considerable improvement over the alternatives. An oblivious assignment, where we assign dividers in each dimension independently, only guarantees a W/p bound, or a p^{d-1} ratio. And the simplest partition – uniform assignment – can be as much as p^d away from optimal.

4 Implementation

We sketch in this section a complexity analysis of the iterative refinement algorithm. The bounds obtained improve on those claimed in the previous papers of Nicol [11] and Manne and Sørevik [9]. To begin with, we generalize a result of Nicol [11] on 1-D solutions to a more general class of cost functions in order to handle the special requirements of the modified algorithms.

4.1 1-D computation

The 1-D case has been considered quite frequently in the literature. Olstad and Manne [12] gave a $O(pn)$ algorithm that holds for all monotone cost functions that are invertive, i.e. where one can add or subtract a single element in constant time. Frederickson [2] studied parameterized range searching, which solves the problem in linear time $O(n)$ for a range of cost functions that includes the standard additive function. In practice, however, the algorithm is likely to be both too slow and complicated. Nicol [11] gave a bottleneck search algorithm, which solves the case of the standard cost function in time $O(n + (p \log n)^2)$.

We describe here how Nicol's approach can be extended to any modular cost function, with no added time complexity.

Lemma 4.1 *Let ϕ be a cost function where $\phi(B_1 \cup B_2)$ can be computed in time t using $\phi(B_1)$ and $\phi(B_2)$. Then, the optimal column partitioning with respect to ϕ can be computed in time $O(t(p \log n)^2)$, given $O(n)$ preprocessing.*

Proof. The preliminary part of our approach is to precompute the cost of segments of size that is a power of two and end at some multiple of that power of two. Namely, we precompute the cost of each segment $[k \cdot 2^{l-1} + 1, (k+1)2^l]$, for each l and k within the range. This allows us to compute the cost of any segment in logarithmic time. This precomputation need only be done once, even if this 1-D algorithm is executed multiple times.

A key subroutine used by the main algorithm is *greedy partition*, which is given a fixed value Q and attempts to partition into p blocks of cost at most Q. This algorithm can be performed in time $p \log n$ when the precomputed values are available. For each block $i = 1, \ldots, p$, the algorithm performs a binary search on the set of remaining values for an index z such that the cost of the segment up to but not including element z is at most the given bottleneck value Q while the cost would exceed Q if the element z is included. This binary search can be performed in such a way that in each step we compute a new segment from a previous segment and a precomputed segment of size power of two. This takes time at most $\log n$, given a modular cost function.

If we have covered all the n elements in the p partitions, then the value Q succeeds; if Q does not succeed, then there is no partitioning of cost at most Q. Hence, if we are given the optimal cost OPT, the greedy partitioning will find an optimal partition. The value OPT must be the cost of some segment in the array, but the issue remains how to find that efficiently.

We then proceed as Nicol. Set *start* to be the starting index, 1. First find an index z such that the cost of $[start, \ldots, z-1]$ is insufficient as bottleneck cost (i.e. that greedy partitioning fails for this value) while the cost of $[start, \ldots, z]$ suffices. We can again use binary search, adding in each step a power-of-two-sized set to a previous segment, to find this in $\log n$ calls to *greedy*. The optimal solution must have $[start, \ldots, z]$ as its first block. We then recurse, for the remaining $p - 1$ blocks, on the array suffix starting at index z. ∎

In particular, the above argument applies to the cost function consisting of the sum of the elements in a block excluding the heaviest, which we used in a segmented form in the second sweep of the 3-sweep algorithm. For this, we store with each computed block the cost of the heaviest element. Note that none of the previously studied algorithms (beyond the trivial ones) treat this case.

4.2 2-D computation

In our 2-D algorithms, we used four different objectives for sweeps:

- The first sweep is the simplest, and amounts to a plain 1-D sweep on a fully preprocessed instance. Time complexity is $(p \log n)^2$ given n^2 preprocessing
- Other sweeps of the pure iterative refinement algorithm involve p segments, for a cost of p per operation. Time complexity is thus $p^3 \log^2 n$.
- The segmented sweep that excludes the cost of the heaviest row segment, as in the second step of the three-sweep algorithm, also takes p time per operation, or $p^3 \log^2 n$ in total.
- Minimizing the cost of a row segment, or a column segment, is equivalent to a segmented sweep when the number of segments is n. Thus, the time complexity is $n(p \log n)^2$. (Note, however, that this could be decreased by considering only rows of high cost.)

It follows that the pure iterative refinement method runs in time $O(n^2 + Tp^3 \log^2 n)$, where T is the number of sweeps. This improves on the bound of $O(n^2 + p^4 \log^2 n)$ obtained by Nicol.

The three sweep variation runs in time $O(n^2 + p^3 \log^2 n)$, while the two sweep method takes time $O(n^2 + n(p \log n)^2)$. The former is linear in the size of the input whenever $p = O((n/\log n)^{2/3})$, while the latter is linear whenever $p \leq \sqrt{n}/\log n$.

The case when p is a constant or a slow-growing function w.r.t. n is of special interest, especially in the applications to load-balancing parallel computers. Then, the time complexity of the algorithm is poly-logarithmic in n, given the $O(n^2)$ preprocessing step (which is e.g. trivially parallelizable.)

References

[1] B. Chapman, P. Mehrotra, and H. Zima, *Extending HPF for advanced data parallel applications*, IEEE Trans. Par. Dist. Syst., (Fall 1994), pp. 59–70.

[2] G. N. Frederickson, *Optimal algorithms for partitioning trees and locating p-centers in in trees*, in Proceedings of second ACM-SIAM Symposium on Discrete Algorithms, 1991, pp. 168–177.

[3] I. P. Goulden and L. B. Richmond, Problem 6386, *American Mathematical Monthly*, May 1982, p. 338.

[4] M. Grigni and F. Manne, *On the complexity of the generalized block distribution*, in Proceedings of Irregular'96, the third international workshop on parallel algorithms for irregularly structured problems, Lecture Notes in Computer Science 1117, Springer, 1996, pp. 319–326.

[5] *High Performance Fortran Language Specification 2.0*, January 1997. Available from http://www.crpc.rice.edu/HPFF/home.html.

[6] L. Holst, *On discrete spacings and the Bose-Einstein distribution*, in Contributions to Probability and Statistics in the Honor of Gunnar Blom, Eds J. Lanke and G. Lindgren, Department of Mathematical Statistics, Lund University, Lund, Sweden 1985 pp. 169–177.

[7] S. Khanna, S. Muthukrishnan, and S. Skiena, *Efficient array partitioning*. Proceedings of the 24th International Colloquium on Para Automata, Languages, and Programming (ICALP), Lecture Notes in Computer Science 1256, Springer, 1997, pp. 616–626.

[8] F. Manne, *Load Balancing in Parallel Sparse Matrix Computations*, PhD thesis, University of Bergen, Norway, 1993.

[9] F. Manne and T. Sørevik, *Partitioning an array onto a mesh of processors*, in Proceedings of '96, Workshop on Applied Parallel Computing in Industrial Problems and Optimization, Lecture Notes in Computer Science 1184, Springer, 1996, pp. 467–477.

[10] A. Mingozzi, S. Ricciardelli, and M. Spadoni, *Partitioning a matrix to minimize the maximum cost*, Disc. Appl. Math., 62 (1995), pp. 221–248.

[11] D. M. Nicol, *Rectilinear partitioning of irregular data parallel computations*, J. Par. Dist. Comp., (1994), pp. 119–134.

[12] B. Olstad and F. Manne, *Efficient partitioning of sequences*, IEEE Trans. Comput., 44 (1995), pp. 1322–1326.

[13] L. Takács, Solution to Problem 6386, *American Mathematical Monthly*, December 1983, p. 710.