Robust Shearsort on Incomplete Bypass Meshes

Behrooz Parhami and Ching Yu Hung

Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

ABSTRACT

An incomplete 2-D bypass mesh is a rectangular m×n grid of p processing elements (PEs) in which faulty PEs have been bypassed in their respective rows and columns. Thus, two PEs that are separated only by faulty PEs in the same row or column become neighbors. We show that given O(log m) PE faults in each column, a robust version of the sorting scheme known as shearsort has the same O(m + n log m) asymptotic time complexity on such an incomplete mesh as on a complete mesh. We also demonstrate how the ability to compute on incomplete meshes is relevant to fault tolerance by presenting reconfiguration schemes to convert meshes with faulty PEs into incomplete bypass meshes.

Keywords: Algorithm complexity, Defect tolerance, Fault tolerance, Mesh with row and column buses, Mesh-connected parallel computers, Parallel sorting, Separable row and column buses, sorting algorithms.

I. INTRODUCTION

A two-dimensional mesh-connected computer, or 2-D MCC, consists of $p = mn$ processing elements (PEs) arranged in an $m \times n$ grid (m rows, n columns), where horizontally and vertically adjacent processors are connected via local communication links (E, W, N, S). Because of their potential for dense VLSI realizations, 2-D MCCs have become popular and numerous researchers have studied their properties pertaining to efficient algorithm implementation and fault tolerance. With the notable exception of adaptive routing algorithms [LIND91], the two issues of algorithm design and fault tolerance are dealt with separately in that algorithm developers assume a complete (fault-free) mesh and fault tolerance techniques aim at restoring such a complete mesh by reconfiguring a faulty one.

With respect to algorithm implementation, researchers have shown how basic building-block computations and data routing primitives can be programmed through efficient (pipelined) use of the E, W, N, S connections [LEIG92], [CYPH94]. It has also been observed that the primary disadvantage of a standard mesh or torus is that the number of steps required by most algorithms is lower-bounded by its diameter; i.e., any non-trivial algorithm has time complexity $\Omega(m + n)$. Consequently, many authors have proposed meshes augmented with broadcast buses and other global communication facilities (e.g., [PRAS87], [SERR93]) or algorithms that take advantage of the less distance-sensitive wormhole routing scheme [KIMD94].

As for fault tolerance, the main focus has been on methods that can salvage a smaller working mesh from a larger one that contains faulty PEs or links [NEGR86], [DAVI94]. Usually, an $m \times n$ mesh with spare rows/columns is converted to an $m' \times n'$ mesh (where $m' \le m$ and $n' \le n$) using embedded switching mechanisms, although certain low-redundancy architectures and more efficient shared-spare schemes have also been considered. The main reason for the focus on restoring a complete mesh has been that both main application areas, viz defect tolerance and systolic numerical computation, have traditionally required the availability of a complete working mesh.

In this paper, we take a different approach and investigate how much fault tolerance can be achieved through regular mechanisms in MCCs such as separable row/column buses, originally added to improve their performance for certain global computations, or bypass mechanisms included to reduce the data routing latency. We do this by focusing on a particular "robust" sorting algorithm (adapted from shearsort [SCHE86], [SCHE89]) and showing that it runs with the same asymptotic time complexity on incomplete bypass meshes as on complete MCCs. Our ultimate goal is to characterize the extent of fault tolerance available and the corresponding performance/dependability tradeoffs.

A key assumption throughout the paper is that a fault-free PE can detect if any of its neighbors has failed and that a link fault is detected by a fault-free PE at either end. Faults are assumed to be permanent and diagnosis off-line. While we do not wish to discount the difficulty of complete and prompt fault detection and diagnosis, a comprehensive treatment of this problem is beyond our scope here [HOSS89], [SOMA92].

The remainder of this paper is organized as follows. In Section II, we consider abstract MCCs in which faulty PEs can be detected and bypassed without worrying about how the bypassing is actually accomplished. We describe the robust shearsort algorithm in Section III and prove its convergence in Section IV. Next, we discuss the worst-case and average-case performance of robust shearsort in Sections V and VI, respectively. Fault tolerance implications of the parallel sorting results and hardware issues are dealt with in Section VII. Section VIII contains our conclusions and recommendations for further research.

II. Incomplete Bypass Meshes

Certain applications do not need the complete connectivity of 2-D MCCs for efficient handling but rather treat the mesh as a logical 1-D array with better communication latency characteristics than a physical 1-D array. For example, if data is to be sorted, it is not essential that a complete $m \times n$ mesh be available. As long as the faulty PEs are detected, disabled, and bypassed, it may be possible to perform the required data comparisons and movements that would eventually lead to sorted order. Many geometric problems on point sets [AKLS93] also fall in this class.

Accordingly, we assume that any faulty PE can be bypassed in its respective row/column by setting certain routing switches. As an example, in the 4×4 MCC containing 3 faulty processors shown in Figure 1, bypassing of the faulty PEs along the rows and columns leads to the configuration of Figure 2. Note that in Figure 2, bypassing is assumed to have occurred only in horizontal or vertical direction. Thus, the faulty PE in the upper-right corner of Figure 1 is not bypassed to connect PEs 2 and 7, even though this would have been quite feasible. The reason is that such horizontal-to-vertical bypass connections add complexity to the PEs and to most algorithms and would thus tend to nullify the gains resulting from shorter graph-theoretic distances.

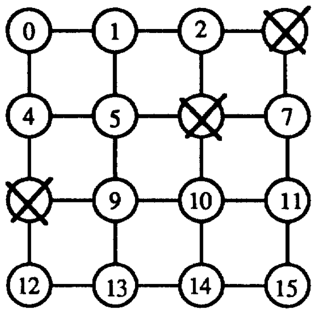

Figure 1. A 2-D mesh-connected computer with 3 faulty processors.

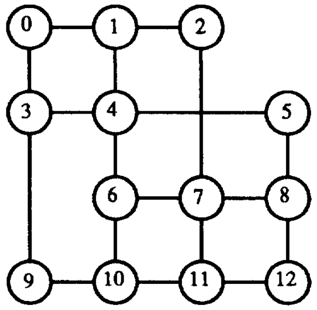

Figure 2. Incomplete bypass mesh corresponding to Figure 1.

In all cases, we will assume that the incomplete mesh remains connected. Thus degenerate cases, such as the one in which processors 1, 3, 4, 6, 9, 11, 12, and 14 in the 4×4 mesh of Figure 1 are faulty, will not be allowed. However, we do not limit the number of faulty PEs a priori. Thus, extreme cases with a large number of faults, such as the one shown in Figure 3, will be allowed. We will see that our sorting algorithm runs correctly on such unlikely configurations, albeit at highly reduced performance. A variant of the incomplete bypass mesh model (Figure 4) leads to higher efficiency for our robust shearsort algorithm. This variant is obtained through our normal bypassing scheme if the original mesh contains "snakelike" edge links. Since this variation does not increase the node complexity, we will use it freely in the remainder of this paper.

It is interesting to note that regardless of the number of bypassed (faulty) PEs, the diameter of an incomplete $m \times n$ bypass mesh does not exceed $m + n - 2$ [PARH94]. This result is non-trivial in view of extreme cases such as the one depicted in Figure 3. Due to the above upper bound, the possibility of efficient $O(m + n)$-time algorithms on incomplete meshes is not ruled out by its diameter. In fact, it can be shown that semigroup and parallel prefix computations can be performed in asymptotically optimal time on such incomplete meshes and that robust data routing algorithms can be devised that suffer virtually no performance degradation compared to those running on a complete mesh [PARH94].

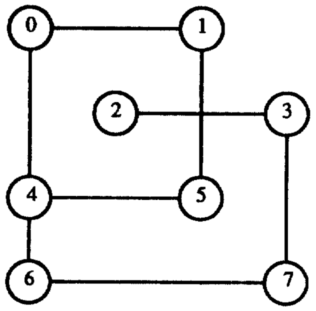

Figure 3. An extreme case of an incomplete bypass mesh.

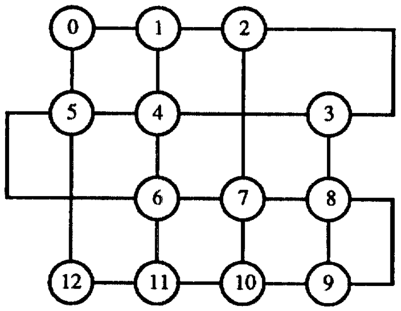

Figure 4. Incomplete bypass mesh with snakelike edge links.

Here we focus on sorting data on incomplete bypass meshes. Clearly, a standard mesh sorting algorithm is inapplicable because all of its required comparisons and swaps cannot be performed directly. An attempt to simulate a complete mesh on an incomplete bypass mesh through load redistribution and establishment of multi-link "virtual channels" among nodes that previously contained neighboring elements leads to excessive overhead and unbounded deterioration of performance for certain fault patterns. Thus, we opt for developing a robust algorithm that would run directly on an incomplete bypass mesh. We assume, for the sake of efficiency, that the incomplete bypass mesh is augmented with "snakelike" edge links, as shown in Figure 4.

III. ROBUST SHEARSORT

The shearsort algorithm [SCHE86], [SCHE89] is an elegant, but suboptimal, sorting algorithms for 2-D MCCs that works on the basis of alternating row and column sorts. Assuming a single data item per PE, the sequence row-sort (in alternating or snakelike order) followed by column-sort (top-to-bottom) is repeated $\lceil \log_2 m \rceil$ times and at the end, a row-sort step, in left-to-right or snakelike order, completes the process. This leads to $O((m + n) \log m)$-time complexity. A more efficient version, that replaces the ith column sort by $m/2^i$ steps of odd-even transposition (slightly more if m is not a power of 2) results in $O(m + n \log m)$ complexity. The original proof of shearsort's convergence and time complexity are based on the zero-one principle [KNUT73] and the fact that with each pair of row/column sorts, half of the remaining "dirty" rows (containing both 0 and 1 elements) are "cleaned up" (contain either only 0 or 1 elements, but not both types). It is easy to see that the proof does not extend to an incomplete bypass mesh in general.

An adaptation of shearsort, dubbed "robust shearsort" for its ability to work in the presence of faulty PEs and links, works on incomplete bypass meshes of the type shown in Figure 4 (i.e., with snakelike edge links). As in ordinary shearsort, rows and columns are alternately sorted without regard to the presence of faulty PEs. In Section IV, using the zero-one principle, we will prove that in $O(\log m)$ steps, this process converges to an almost sorted array of 0s and 1s, the unsorted portion being confined to a subarray of small height h, where h is related to the maximum number of faults in any given column. This $h \times n$ subarray is then sorted by odd-even transposition along the snake spanning the entire mesh in time $O(hn)$. If h is $O(\log m)$, the overall $O(m + n \log m)$ complexity of Shearsort is preserved.

The exact worst-case complexity of optimized shearsort is $2m + n(\log_2 m + 1) + \log_2 m$. Thus, for a given number $p = mn$ of processors, the optimal mesh architecture with regard to optimized shearsort is non-square, with m selected to satisfy $p \approx m^2/\log_2 m$. Similarly, robust shearsort has worst-case time complexity $2m + n(\log_2 m + h) + h \log_2 m$ (see Section V) and has a corresponding optimal non-square architecture whose aspect ratio depends on the expected number of faults. When no fault exists in the processor array, we have $h = 1$ and robust shearsort performs exactly as ordinary shearsort. With $O(1)$ faults in each column, we have $h = O(1)$ and robust shearsort is only slightly slower than ordinary shearsort. The slowdown factor remains bounded by a small constant as long as there are $O(\log m)$ faults in each column, leading to $h = O(\log m)$.

Thus, robust shearsort is very efficient in the most common cases and offers graceful degradation as the number of faults grows. With a constant failure rate, the number of faults in each column, and thus h, is actually $O(m)$. With $O(m)$ faults, the time complexity of robust shearsort deteriorates to $O(mn)$. However, this drawback is only of theoretical significance in view of the extremely small constants involved.

IV. CONVERGENCE OF ROBUST SHEARSORT

In this section, we look at one iteration of shearsort, consisting of a row sort step and the subsequent column sort step. Using the zero-one principle, we demonstrate how the number of dirty rows is reduced from d to d' in one iteration (and thus from m to some small quantity h over many iterations), hence establishing the convergence and allowing us to deduce the worst-case performance in Section V.

For now, let the number d of dirty rows before the iteration be even. The row sort does not affect the number of dirty rows so it remains d right before the column sort. Let us consider only the d dirty rows of the $m \times n$ mesh, numbered 0 to $d - 1$, for the moment. This $d \times n$ dirty submesh is partitioned into two planes. The even plane consisting of all even-numbered rows and the odd plane consisting of all odd-numbered rows; each is of size $d/2 \times n$.

We define $f_1(i)$ as the number of 1s in Column i of the even plane and $g_1(i)$ as the number of 1s in Column i of the odd plane. Similarly $f_x(i)$ and $g_x(i)$ are defined as the number of faulty PEs in Column i of the even and odd planes, respectively.

When there is no fault in the $d \times n$ submesh, after the row sort according to snake-like ordering, $f_1(i)$ is a nondecreasing function of i and $g_1(i)$ is a nonincreasing function of i. These are also true after the column sort, since the column sort does not change the count of 1s in any column. With faulty PEs, the above may not hold. However, since it is always possible to replace all Xs (values in faulty PEs) with 0s and 1s in such a way that the sorted order of rows is preserved, there must exist functions $f'_x(i)$ and $g'_x(i)$, satisfying $0 \le f'_x(i) \le f_x(i)$ and $0 \le g'_x(i) \le g_x(i)$, such that $f_1(i) + f'_x(i)$ is a nondecreasing function of i, and $g_1(i) + g'_x(i)$ is a nonincreasing function of i. Intuitively, f' and g' denote the number of Xs that were replaced by 1s in the even and odd planes, respectively.

These observations lead to the following result.

Lemma 1: With some faulty PEs present in the submesh, we have for $a \le b, f_1(a) \le f_1(b) + f_x(b)$ and $g_1(b) \le g_1(a) + g_x(a)$.

Proof: Immediate from the four inequalities $f_1(a) + f'_x(a) \le f_1(b) + f'_x(b)$, $f'_x(b) \le f_x(b)$, $g_1(b) + g'_x(b) \le g_1(a) + g'_x(a)$, and $g'_x(a) \le g_x(a)$. ▮

The following lemma relates the number d' of dirty rows remaining after the column sort to $f_1(i)$ and $g_1(i)$ that we have defined.

Lemma 2: The number of clean 1 rows after the column sort is at least $\min_i[f_1(i) + g_1(i)]$. The number of clean 0 rows after the column sort is at least $d - \max_i[f_1(i) + f_x(i) + g_1(i) + g_x(i)]$. Consequently, the number d' of dirty rows remaining after an iteration satisfies $d' \le \max_i[f_1(i) + f_x(i) + g_1(i) + g_x(i)] - \min_i[f_1(i) + g_1(i)]$.

Proof: The first statement clearly holds true; $\min_i[f_1(i) + g_1(i)]$ is the minimal number of 1s present in any column. If we have no fault in the last $\min_i[f_1(i) + g_1(i)]$ rows, then there would be exactly this many clean 1 rows. If there are faulty PEs in these rows, 1s in some of the columns stack higher and we may end up with more clean 1 rows. Similarly, the number of clean 0 rows is derived as $\min_i[(d/2 - f_1(i) - f_x(i)) + (d/2 - g_1(i) - g_x(i))] = d - \max_i[f_1(i) + f_x(i) + g_1(i) + g_x(i)]$. The third statement follows from the first two. ▮

Theorem 1: Let $x = \max(\max_i f_x(i), \max_i g_x(i))$ be the maximum number of faults in one column of the even or odd plane. Then, the number d' of dirty rows remaining after one iteration of shearsort is related to the initial number d of dirty rows by $d' \le d/2 + 2x$.

Proof: Assume that the maximum of $f_1(i) + f_x(i) + g_1(i) + g_x(i)$ occurs in Column a and the minimum of $f_1(i) + g_1(i))$ occurs in Column b. There are two cases to consider:

Case 1: $a \le b$. By Lemma 1, $f_1(a) - f_1(b) \le x$. In addition, $g_1(a) + g_x(a)$ is the number of nonzeros in Column a, so $0 \le g_1(a) + g_x(a) \le d/2$, and similarly $0 \le g_1(b) \le d/2$. Their difference $g_1(a) + g_x(a) - g_1(b)$ is thus at most $d/2$.

Case 2: $a > b$. By Lemma 1, we have $g_1(a) - g_1(b) \leq x$. Also we have $f_1(a) + f_x(a) - f_1(b) \leq d/2$.

In either case, the theorem follows. ▮

The following example illustrates that the bound given in Theorem 1 for the number of remaining dirty rows after one iteration is tight. Here we have $d = 6$, $x = 1$, and $d' = 5$.

After row sort	After column sort
0 0 0 1	0 0 0 0
1 1 0 0	0 1 0 0
0 X 0 1	0 X 0 0
1 1 0 0	1 1 0 1
0 1 X 1	1 1 X 1
1 1 0 0	1 1 0 1

Theorem 1 supplies information on the number of clean rows gained after each iteration of Shearsort. If x is small compared to d, say $x = O(1)$, then the number of dirty rows is roughly halved with each iteration. On the other hand, when $x \geq d/4$, the number of dirty rows may remain unchanged. Thus, as discussed in Section III, some other sorting method, e.g. odd-even transposition sort, has to take over. In this case, $d \times n$ is the required number of odd-even transposition steps along the overall snake needed to complete the sorting. Hence, the proof of convergence is complete.

V. WORST-CASE PERFORMANCE

Armed with the results of Section IV, we can now analyze the worst-case performance of robust shearsort. We will see in Section VI that the worst-case analyses are highly pessimistic and that simulation results predict much better average-case performance with randomly distributed faults. In the analyses that follow, both k and $\log_2 m$ stand for the integral value $\lceil \log_2 m \rceil$.

Relaxing the assumption that d is even leads to the recurrence $d_{i+1} \leq (d_i + 1)/2 + 2x$ instead of the inequality stated and proved in Theorem 1. This leads to the condition $d > 4x + 1$, instead of $d > 4x$, for achieving a guaranteed reduction in the number of dirty rows.

Lemma 3: The condition $d \leq 4x + 1$ is guaranteed to hold after $\log_2 m$ iterations of row-column sorts.

Proof: We can write:

$$d_{i+1} \leq (d_i + 1)/2 + 2x = d_i/2 + (4x + 1)/2$$

Unfolding the recurrence, we get:

$$d_k \leq d_{k-1}/2 + (4x + 1)/2 \leq \ldots$$
$$\leq d_0/2^k + (1/2^k + 1/2^{k-1} + \ldots + 1/2)(4x + 1)$$

Since $d_0/2^k = m/2^{\log_2 m} \leq 1$ and the coefficient of $4x+1$ on the right-hand side is strictly less than 1, we must have $d_k \leq 4x + 1$. ▮

Theorem 2: Robust shearsort has worst-case complexities $m \log_2 m + n(\log_2 m + 4x + 1)$ and $2m + n(\log_2 m + 4x + 1) + (4x + 1)(\log_2 m - 1)$ for its simple and optimized versions, respectively.

Proof: The time complexity of the simple version of shearsort is obtained by observing that each of the $\log_2 m$ pairs of row/column sorts takes $m + n$ steps and that odd-even transposition sorting of the remaining $4x + 1$ dirty rows takes $(4x + 1)n$ additional steps. Just as in optimized shearsort, the column sort in the ith iteration requires at most d_i steps of odd-even transposition. From the recurrence $d_{i+1} \leq (d_i + 1)/2 + 2x$ and the initial condition $d_0 = m$, we obtain the upper bound

$$\begin{aligned}\sum_{i=0}^{k-1} d_i &\leq m + [(m + 1)/2 + 2x] + [(m + 3)/4 + 3x] \\ &\quad + [(m + 7)/8 + 7x/2] + \ldots \\ &\quad + [(m + 2^{k-1} - 1)/2^{k-1} + 4x - x/2^{k-3}] \\ &= [m + m/2 + \ldots + m/2^{k-1}] + [1/2 + 3/4 \\ &\quad + \ldots + (2^{k-1} - 1)/2^{k-1}] + \sum_{i=0}^{k-1} (4x - x/2^{i-2}) \\ &< 2m + (4x + 1)(\log_2 m - 1)\end{aligned}$$

for the total number of steps during column sorts. Including the $\log_2 m$ row sorts and the final odd-even transposition sort of length at most $(4x + 1)n$, the total complexity is upper bounded by $2m + n(\log_2 m + 4x + 1) + (4x + 1)(\log_2 m - 1)$. ▮

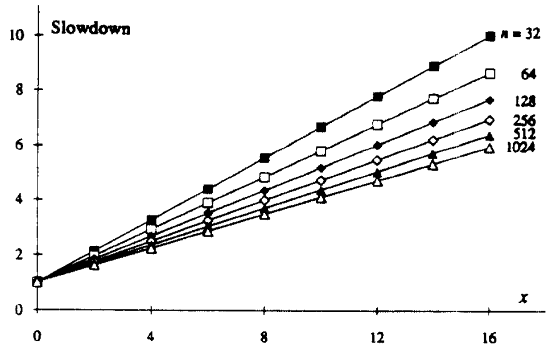

Figure 5. Worst-case slowdown factor of robust shearsort as a function of the maximum number x of faults in a column of the odd or even plane.

The upper bound on relative slowdown of robust shearsort on square $n \times n$ meshes of different sizes, and with various maximum number of column faults, is plotted in Figure 5. Recall that x is the maximum number of faults in the odd-numbered or even-numbered elements of a column. Hence, for example, $x = 16$ represents an unrealistically large number of faults for the smaller 32×32 or 64×64 meshes considered. In the more realistic region, the worst-case slowdown is much smaller (typically below 2 or 3). The average-case performance is even better (see Section VI).

Thus far, we have ignored the effect of the reduction of the number of PEs from $p = mn$ to p' due to faults. Essentially, we have assumed that once faults occur, the problem size is reduced from p to p'. However, to be fair in our performance comparisons, we must keep the problem size constant. In practice, each PE in the complete mesh may have been assigned a "load" of N/p data items. After an initial sorting phase of length $O(N/p \log(N/p))$ locally within each PE, mesh sorting would then be performed with a slowdown factor of N/p compared to a mesh of size N, as each PE successively "emulates" each of the N/p "virtual" PEs assigned to it.

If faults leave us with only p' PEs, each PE must deal with the higher load of N/p' data items. Thus, on top of the slowdown resulting from modified connectivity as discussed before, the initial sorting takes $O(N/p' \log(N/p'))$ steps and a slowdown factor of p/p', relating to the increased load, will be experienced in the subsequent mesh sorting phase.

Denoting the fraction $(p - p')/p$ of faulty or otherwise unusable PEs by ϕ, the latter slowdown factor becomes $1/(1 - \phi)$, which is $1 + \phi/(1 - \phi)$, or approximately $1 + \phi$ if ϕ is small. The former slowdown factor $(p/p') \log(N/p')/\log(N/p) \approx (1 + \phi)(1 - \log(1 - \phi)/\log(N/p) \approx (1 + \phi)(1 + \phi/\log(N/p))$ is only marginally higher than $1 + \phi$. For large meshes, ϕ is unlikely to exceed 0.01 (say). Thus, these additional slowdowns are not worrisome.

VI. AVERAGE-CASE PERFORMANCE

Note that x is associated with faulty processors in the dirty rows, not in the entire mesh. Suppose there are $m/10$ faulty PEs in a column. The analysis offered in Section IV is based on the highly pessimistic assumption that $x = m/10$ throughout the required $\log_2 m$ iterations, leading to up to $4m/10 + 1$ dirty rows at the end. However, x will become smaller and smaller in the latter part of the iterations and it is very likely that the algorithm converges to $O(1)$ dirty rows; it converges to $O(m)$ dirty rows only if most of the faults are clustered tightly together, which is very unlikely. Moreover, faults that are near the top and the bottom of the dirty subarray might increase the number of clean rows. Thus, when x is comparable to d, the bounds in Theorem 2 are pessimistic with high probability.

Before presenting our results on the average-case behavior, we discuss an improvement of robust shearsort that can be incorporated into the algorithm with some computation at system reconfiguration time. Whenever a new fault is detected, the system enters a reconfiguration phase in which the appropriate bypass connections are established and fault information is propagated. We assume that this phase includes a global computation that establishes the successive number of dirty rows in the worst case based on the knowledge of the fault pattern. This information is then used to guide the number of iterations and the number of odd-even transposition steps carried out during the column sort steps. We stress that this improvement is relatively minor and that much of what follows applies, with small adjustments, to the version discussed earlier.

To get a feel for the average-case behavior of robust shearsort, we have developed a simulation program that given a mesh size and the total number f of faults (within a range), simulates the robust shearsort algorithm (improved as discussed in the preceding paragraph) and returns the distribution of the number of steps needed to sort the data with a prespecified number of runs.

As an example, using 10,000 runs on a 32×32 mesh having f randomly distributed faults (for $f \in \{2, 4, 8, 16, 32\}$), we have obtained the results depicted in Figure 6. We observe from Figure 6 that for up to 16 faults, the expected slowdown factor does not exceed 2 and that even with 32 faults, it remains well below 2.5.

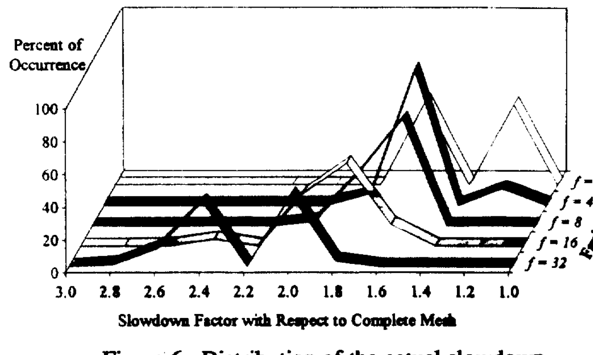

Figure 6. Distribution of the actual slowdown factor for robust shearsort as a function of the total number f of faults in a 32×32 mesh.

VII. FAULT TOLERANCE IMPLICATIONS

In this section, we will discuss a way to assure that faulty PEs can be bypassed using broadcast mechanisms, typically provided for improved performance in MCCs. Numerous other ways can be contemplated to implement the bypass links.

A separable row/column bus is a broadcast mechanism to which all PEs in a particular row/column are connected and which can be sectioned into several local buses via PE-controlled switches. If a switch is inserted after each PE, then any sectioning pattern is realizable. This is the ideal case with regard to our fault tolerance scheme. If a switch were placed after a block of l PEs on each row/column, an entire block of l PEs would have to be bypassed as a result of a single PE failure. This would lead to poor performance for robust shearsort as the parameter x, used in the analyses of Section V, could become large with only a few actual faults.

Consider the column buses and assume that each PE can control the two switches to its north and south (Figure 7). To establish vertical bypass connections going north, a fault-free PE will open the switch to its south if its north neighbor is faulty. Similarly, the north switch is opened if the south neighbor is faulty. An obvious limitation is that both neighbors should not be faulty at the same time, since it is impossible for a PE to send to the north and receive from the south on the same bus. This isn't a serious drawback, since a PE surrounded by two faulty PEs can be assumed to have failed. When a mesh algorithm calls for communication with the north neighbor, a sender (receiver) whose north (south) neighbor is faulty, puts (gets) the data on (from) the column bus.

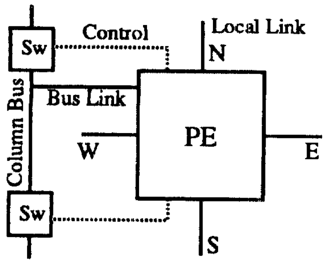

Figure 7. Configuration of separable row/column buses to provide bypass links.

The scheme proposed above is somewhat inefficient in that it involves separate connections (local links and separable buses) that are never used simultaneously under fault-free and fault conditions and requires each PE to control 4 switches (2 in each dimension).

Figure 8 shows an alternate architecture that merges the separable buses with local connections, thus reducing the number of ports per PE from 6 to 4 and the number of switches controlled by each PE from 4 to 2. When no fault is present, all switches are opened to allow local N-S and E-W communications. With faults present, a scheme similar to that discussed earlier can be used to bypass faulty PEs horizontally or vertically. The main point is that no speed is lost due to the shared links because their usage was mutually exclusive.

A distinct advantage of this new architecture over the previous one is that even an isolated PE, surrounded on all sides by faulty PEs, can be effectively utilized. This is due to the fact that each PE has two connections to the row and column buses and can thus send and receive on the same bus when its associated bus switch is open.

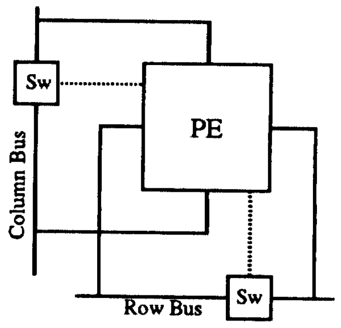

Figure 8. Alternate configuration for separable row/column buses.

An issue of concern is whether the bypass connections realized through row/column buses or the alternate architecture discussed above can be operated at the same speed as local, nearest-neighbor connections (otherwise an additional slowdown will be introduced). Unlike row/column broadcasts, which may be slowed down by the large number of switches on a given bus, the delay through a bypass connection depends only on the number of switches between the source and destination PEs which is the same as the grid distance between the two. Even with highly pessimistic failure probabilities, it is virtually impossible to have such large failure clusters as to render the bypass delay unacceptable. Besides, one can always bypass an entire row or column if long failure chains do in fact occur.

VIII. CONCLUSIONS

By adapting the shearsort algorithm to run on an incomplete bypass mesh, we have shown that certain computations can be performed efficiently, and with graceful degradation, on faulty meshes modeled in this way. This result, combined with similar algorithms for other building-block computations (reduction, parallel prefix, data routing), pave the way for implementing more complex algorithms on incomplete meshes and lead to a new fault tolerance strategy based on robust algorithms. We have also demonstrated how the logical bypass connections assumed in our model can be physically realized through the use of separable row and column buses.

Many interesting problems arise in the study of such robust algorithms in connection with MCCs and other architectures. For example, in adapting the shearsort algorithm, we made no assumption about the distribution of faults. It is easy to prove that with a constant failure rate, the existence of $n/2$ complete columns in a faulty $m \times n$ mesh is virtually guaranteed as long as the mesh is not too small or too large. This observation may lead to efficient sorting algorithms or to higher performance for a more clever adaptation of shearsort. At this point, it is not clear whether or how other more efficient $O(m + n)$-time sorting algorithms can be adapted to run on incomplete bypass meshes.

Many variations are possible. The underlying model can be changed. For example a mesh in which faulty PEs and links create discontinuities or "holes" can be considered in lieu of the bypass model assumed here. The robust design style can be combined with switch-based reconfiguration schemes and provision of spare units in an effort to avoid any degradation as a result of the first few faults, thus resorting to a degraded mode of operation only when the fault tolerance capacity of the reconfigurable architecture is exhausted.

REFERENCES

[AKLS93] Akl, S.G. and K.A. Lyons, *Parallel Computational Geometry*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[CYPH94] Cypher, R. and J.L.C. Sanz, *The SIMD Model of Parallel Computation*, Springer-Verlag, New York, 1994.

[DAVI94] Davis, N.J., F.G. Gray, J.A. Wegner, S.E. Lawson, V. Murthy, and T.S. White, "Reconfiguring Fault-Tolerant Two-Dimensional Array Architectures", *IEEE Micro*, Vol. 14, No. 2, pp. 60-69, Apr. 1994.

[HOSS89] Hosseini, S.H., "On Fault-Tolerant Structure, Distributed Fault-Diagnosis, Reconfiguration, and Recovery of the Array Processors", *IEEE Trans. Computers*, Vol. 38, No. 7, pp. 932-942, July 1989.

[KIMD94] Kim, D. and S.-H. Kim, "$O(\log n)$ Numerical Algorithms on a Mesh with Wormhole Routing", *Information Processing Letters*, Vol. 50, pp. 129-136, 1994.

[KNUT73] Knuth, D.E., *The Art of Computer Programming-- Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[LEIG92] Leighton, F.T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan-Kaufman, 1992.

[LIND91] Linder, D.H. and J.C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes", *IEEE Trans. Computers*, Vol. 40, pp. 2-12, Jan. 1991.

[MAEB90] Maeba, T., S. Tatsumi, and M. Sugaya, "Algorithms for Finding Maximum and Selecting Median on a Processor Array with Separable Global Buses", *Electronics & Commun. in Japan*, Part 3, Vol. 73, No. 6, pp. 39-47, 1990.

[NEGR86] Negrini, R., M. Sami, and R. Stefanelli, "Fault Tolerance Techniques for Array Structures Used in Supercomputing," *Computer*, Vol. 19, pp. 78-87, Feb. 1986.

[PARH93] Parhami, B., "Fault Tolerance Properties of Mesh-Connected Parallel Computers with Separable Row/Column Buses", *Proc. 36th Midwest Symp. on Circuits and Systems*, Detroit, MI, Aug. 1993, pp. 1128-1131.

[PARH94] Parhami, B. and C.Y. Hung, "Parallel Computation on Incomplete Meshes", revised for publication in *Journal of Computer and Software Engineering*.

[PRAS87] Prasanna-Kumar, V.K. and C.S. Raghavendra, "Array processor with multiple broadcasting", *Journal of Parallel and Distributed Computing*, Vol. 2, pp. 173-190, 1987.

[SCHE86] Scherson, I.D., S. Sen, and A. Shamir, "Shear Sort: A True Two-Dimensional Sorting Technique for VLSI Networks," *Proc. Int'l Conf. Parallel Processing*, 1986, pp. 903-908.

[SCHE89] Scherson, I.D. and S. Sen, "Parallel Sorting in Two-Dimensional VLSI Models of Computation", *IEEE Trans. Computers*, Vol. 38, pp. 238-249, Feb. 1989.

[SERR93] Serrano, M.J. and B. Parhami, "Optimal Architectures and Algorithms for Mesh-Connected Computers with Separable Row/Column Buses", *IEEE Trans. Parallel and Distributed Systems*, Vol. 4, No. 10, pp. 1073-1080, Oct. 1993.

[SOMA92] Somani, A.K. and V.K. Agarwal, "A Distributed Diagnosis Algorithm for Regular Interconnected Structures", *IEEE Trans. Computers*, Vol. 41, pp. 899-906, July 1992.