Yield Enhancement for 3D-Stacked Memory by Redundancy Sharing across Dies

Li Jiang, Rong Ye and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {ljiang,rye,qxu}@cse.cuhk.edu.hk

ABSTRACT

Three-dimensional (3D) memory products are emerging to fulfill the ever-increasing demands of storage capacity. In 3D-stacked memory, redundancy sharing between neighboring vertical memory blocks using short through-silicon vias (TSVs) is a promising solution for yield enhancement. Since different memory dies are with distinct fault bitmaps, how to selectively matching them together to maximize the yield for the bonded 3D-stacked memory is an interesting and relevant problem. In this paper, we present novel solutions to tackle the above problem. Experimental results show that the proposed methodology can significantly increase memory yield when compared to the case that we only bond self-reparable dies together.

1. INTRODUCTION

Three-dimensional (3D) technology, being able to provide abundant interconnect resources with improved performance and less communication energy by integrating multiple silicon dies with short and dense through-silicon vias (TSVs), has become a promising solution to address the unlimited demands for integration capabilities [3]. In particular, to fulfill the ever-increasing demands of storage capacity, many semiconductor companies have implemented 3D-stacked memories, by using TSVs as vertical bus across multiple DRAM layers and it is believed that such memory products will be commercialized in the near future [7, 10].

Due to their extreme high density, memory circuits are prone to manufacturing defects. Consequently, to avoid yield loss, redundant rows and columns are added on-chip so that most faults can be repaired by replacing the rows/columns containing faulty bits with redundant ones. Since we can only implement limited amount of redundant resources on-chip for cost considerations and we cannot afford lengthy repair time for throughput considerations, numerous redundancy analysis and repair algorithms have been presented in the literature for effective and efficient memory repair [2].

Memory circuits typically contain multiple memory blocks and spare rows/columns are added to each memory block. If one of the blocks is not reparable, the entire memory circuit has to be discarded. Intuitively, we can increase memory yield by letting neighboring blocks share the precious redundant rows/columns [15], but this strategy involves quite high routing overhead and hence is not used in practice for traditional two-dimensional (2D) memory circuits. With the emerging 3D-stacked memory, however, sharing redundant resources between neighboring vertical blocks for yield enhancement becomes feasible because short TSVs can be used for routing.

With the above redundancy sharing strategy, a memory block that is not self-reparable can borrow spare resources from its vertical neighbors (if any) and possibly becomes reparable after bonding. Consequently, when compared to the case that we only bond self-reparable dies together to form the 3D-stacked memory, we have the opportunity to achieve significant yield enhancement, especially when the defect density is high and/or the redundant resources are limited. At the same time, whether we could realize this opportunity highly depends on the matching strategy for the memory dies. That is, if a self-reparable memory die is bonded with a non-reparable one but they could not form a functional 3D-stacked memory circuit, the memory yield might even be sacrificed. Therefore, with the distinct defect bitmaps of different memory dies obtained with pre-bond testing, how to selectively matching them together to maximize the yield for the bonded 3D-stacked memory is an interesting and relevant problem. In this paper, we present novel solutions to tackle the above problem. Experimental results with various memory organizations and defect distributions show that the proposed methodology is quite effective for yield enhancement.

The remainder of this paper is organized as follows. Section 2 reviews related work and motivates this paper. The 3D-stacked memory architecture that supports redundancy sharing across neighboring dies is described in Section 3. Next, we discuss the memory repair strategy with redundancy sharing and our memory die matching algorithms in Section 4 and Section 5, respectively. Section 6 presents experimental results with various memory organizations and defect distributions. Finally, Section 7 concludes this paper.

2. PRELIMINARIES AND MOTIVATION

2.1 Prior work

In this subsection, we briefly discuss related work in memory repair and the bonding strategies in 3D circuits.

2.1.1 Memory Repair

The memory repair problem can be formulated as a constrained vertex cover sets of bipartite graphs problem and has been proved to be NP-complete in [9]. It is possible to obtain optimal repair solution by exhaustive search, but this is too time-consuming to be used in practice. To address this problem, various redundancy analysis and fast memory repair techniques (e.g., [4, 9]) were presented in the literature to reduce repair time at the cost of slight memory yield loss.

In the above memory repair strategies, they assume a full fault bitmap of the memory is available and the repair is conducted by external memory testers. With the increasing usage of embedded memories, built-in self-repair (BISR) has become more popular. As it is not cost-effective to store the entire fault bitmap before repair, various memory repair techniques with limited fault bitmap are developed in recent years. In particular, a so-called *essential spare pivoting algorithm* (*ESP*) is presented in [5]. In this work, the authors classify faults into three types: 1) suitable for row repair; 2) suitable for column repair; 3) orthogonal fault which can either be repaired by a spare row or a spare column. This work is shown to be quite effective and efficient.

A memory circuit typically contains multiple memory blocks. If one of these memory blocks is irreparable, the entire memory circuit is deemed as defective and has to be discarded. Clearly, if a self-irreparable memory block can borrow some redundant resources from other reparable blocks, memory yield can be increased. Motivated by the above, [15] proposed a distributed global replaceable redundancy scheme, which allows to use the spare rows/columns in a memory block to repair faults in other memory blocks. In [1], the author proposed a memory built-in self-repair (BISR) algorithm with sharable redundant resources, using a programable decoder. However, such decoder design is much more complex than conventional one, which not only increases area overhead but also significantly deteriorates the routability of memory. Because of this, redundancy sharing is not utilized in practice for today's 2D memory devices.

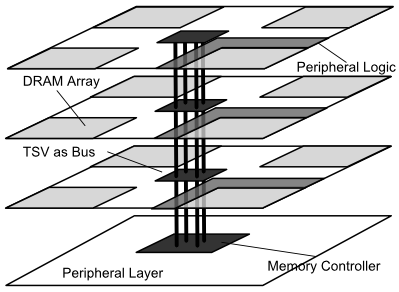

Figure 1: 3D-Stacked DRAM

2.1.2 Bonding Strategies in 3D ICs

3D ICs can be built by stacking multiple silicon layers in several manners: wafer-to-wafer (W2W) bonding, die-to-wafer (D2W) bonding (for 3D ICs built on two semiconductor wafers), or die-to-die (D2D) bonding. The main advantage of W2W bonding is the simplicity of the manufacturing process. However, the yield of the 3D chips can be quite low without "known good die" information [3]. To mitigate this problem, [12] proposed several wafer assignment algorithms that selectively bond wafers together for yield enhancement, assuming that individual dies are tested on-wafer before bonding. Although helpful, bonding wafters together has the instinctive weakness that some good dies are bonded to bad ones and have to be discarded, and hence the manufacturing yield can be still unsatisfactory, especially when the die size is large and/or the defect density is high. D2W/D2D bonding, on the other hand, requires a more complex manufacturing process. However, as we are able to attach known good dies, the manufacturing yield can be significantly higher when compared to W2W bonding [8]. In this work, we assume D2D bonding is utilized to form the 3D-stacked memory chips.

2.2 Motivation

In 3D-stacked memory, bit-arrays are stacked vertically on each other and TSVs are utilized as buses to link the stacked dies together, as shown in Fig. 1. Such organization provides us the opportunity to conduct redundancy sharing across neighboring dies with short TSVs without incurring much routing overhead. With redundancy sharing, an irreparable memory die is likely to become reparable by borrowing spares from its neighboring dies, but whether we can realize the above yield enhancement opportunity highly depends on the matching strategy.

Consider the eight memory dies shown in Fig. 2, each of them containing four memory blocks with different fault density. For simplicity, we classify these dies into four categories. The memory blocks in Type A are all self-repairable and provide extra spare resources (see A1 and A2 in Fig. 2(a)). Type B memory dies cannot repair themselves but become reparable by borrowing a small number of redundant resources from other dies, and we assume that two Type B memory dies stacking together can be reparable. Type C memory dies must borrow plenty of spares to become reparable, while the memory blocks in type D have high fault density and is irreparable even if they are allowed to borrow spares from others.

For this example, suppose we only allow bond self-reparable memory dies only (see Fig. 2(a)), the yield is only 25%. As shown in Fig. 2(b), stacking A1 to B2 and B1 to A2 produces two reparable 3D-stacked memory circuits, and the overall yield in this case is 50%. Suppose we have an aggressive repair strategy as shown in Fig. 2(c), which attempts to repair Type D memory dies using the spare resources in Type A dies, the yield can be 0% in the extreme case. In other words, a bad matching strategy can even reduce the overall memory yield since good dies might be wasted. A good matching strategy as shown in Fig. 2(d), on the other hand, results in a maximum yield of 75%.

From the above, we can conclude that die matching strategy is critical for the final yield of the 3D-stacked memory circuits. Suppose we can quickly know whether the matched 3D-stacked memory is reparable or not for any pair of memory dies, the matching problem is quite similar to the wafer-to-wafer bonding selection problem in [12]. However, with both redundant rows and columns for a memory block, we cannot afford the time used to temporarily running repair algorithm between every possible memory die pairs. Consequently, how to conduct efficient and effective matching to achieve the maximum memory yield is a challenging problem, which motivates this work.

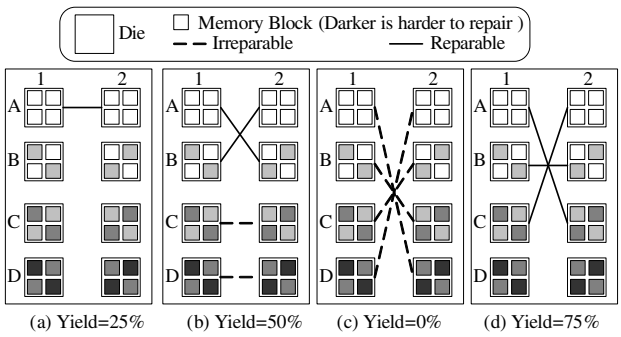

Figure 2: Different Matching Strategy affects the Overall Yield.

3. 3D-STACKED MEMORY ARCHITECTURE WITH REDUNDANCY SHARING

Fig. 3 depicts the 3D-stacked memory architecture that supports redundancy sharing across dies. Spare rows/columns on each memory die not only connects to the programmable decoder on its own layer, but also link to the decoder of other layers using TSVs. The routing overhead to support redundancy sharing across dies is quite low due to the use of short TSVs as routing mechanisms. The pre-fabricated multiplexor controls which memory block use the corresponding spare row/column. For a memory block with n spare rows and m spare columns, sharing all the redundant resources between two neighboring dies requires $n+m$ TSVs, as shown in Fig. 3(a) (only spare rows are shown in this figure). With the continuing improvement of TSV manufacturing technology, this overhead is of a less concern. Moreover, we can leave some spare resources for repairing its own die and/or share the same TSVs for different redundant rows/columns, reducing the amount of TSVs used for redundancy sharing. An example is shown in Fig. 3(b).

The above architecture can tolerate some TSV defects. Let us consider memory blocks in Fig. 3 (a). If one of the three TSVs is defective, we can leave the corresponding spare row to repair its own memory block, while we still have two spare rows for sharing among neighboring blocks. As long as a memory block does not require to borrow all three redundant rows to become reparable, it is still sufficient.

4. MEMORY REPAIR WITH REDUNDANCY SHARING

When conducting self-repair of a memory block, there might be multiple repair strategies to replace the faulty rows/columns with redundant ones, and any one of them suffices. With resource sharing between neighboring dies, however, how to repair each individual die determines whether the other die can be repaired or not. For example, two memory blocks are shown in Fig. 4(a)-(b), in which black vertices denote faulty cells and solid/dashed lines denote spare columns/rows, respectively. Memory block in Fig. 4 (a) is self-irreparable. Memory block in Fig. 4 (b) is self-reparable, but it can lend no spares to the other block if we guarantee to repair itself first. Bonding these two dies together is reparable only if memory block in Fig. 4(a) repairs itself with three spare columns (one is borrowed) while memory block in Fig. 4(b) repairs itself with two spare rows (one is borrowed) and one spare column. From this example, the memory repair algorithm with redundancy sharing must be aware of the fault information of stacked dies and be conducted at a global level.

4.1 Problem Formulation

The problem of memory repair with redundancy sharing can be formulated as follows:

Problem: Given

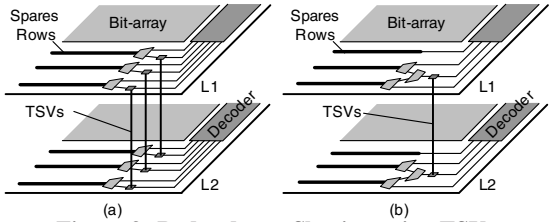

Figure 3: Redundancy Sharing using TSVs.

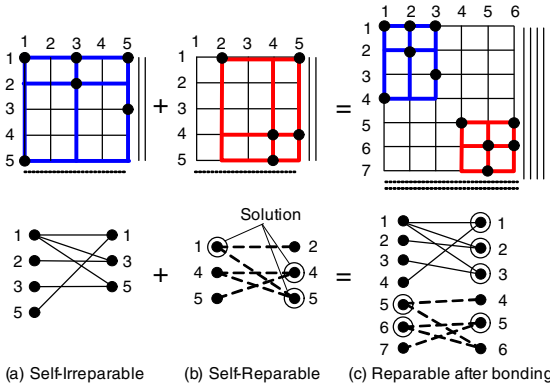

Figure 4: Irrespective Sub-Bipartite Graphs.

- The number of memory blocks N_b on the memory die;
- The fault bitmap of each memory block obtained from pre-bond testing;
- The number of spare rows/columns R_a, C_a in each memory block;

Our objective is to determine the repair scheme of each memory block so that the stacked memory dies can be reparable, whenever possible.

4.2 Memory Repair Strategy

A single memory repair problem is formulated as constrained vertex cover sets of bipartite graphs. Based on the fault bitmap of the memory block, we can build corresponding bipartite graphs (see Fig. 4). The left set of vertices denote the row number and the right set denote the column number of the faulty bits, respectively. An edge between two vertices represents the position of the corresponding faulty bit. With given number of spare rows and columns as constraints, our objective is to find a set of vertices to cover one end of every edge.

We propose a novel algorithm to tackle this problem, inspired from the concept of so-called *irrespective error matrixes* in [1]. We first extract rows and columns that contain faulty bits from the two memory blocks with sharable redundancies, and integrate them together into a new fault bitmap by putting them in diagonal position without intersection, as shown in Fig. 4(c). For the new fault bitmap, our redundant resources are also doubled. By translating the problem of repairing two dies with redundancy sharing into a new problem as above, we guarantee that a sharable redundant row/column can be flexibly utilized to replace a faulty one in either of the two memory blocks without enumerating all possible repair solutions for each die. Then, we can apply any redundancy analysis and repair algorithm to solve this problem. In this example (see Fig. 4(c)), a solution is found so that a self-irreparable block and a self-reparable one forms a reparable stacked memory circuit.

5. MATCHING FOR YIELD ENHANCEMENT

5.1 Problem Formulation

As discussed earlier, how to conduct efficient and effective die matching has a significant impact on the final yield of the 3D-stacked memory circuits. We model this problem by constructing a graph with these memory dies as vertices and add edges between two vertices if the corresponding memory dies are reparable after stacking. This problem can then be solved in polynomial time as the classical *maximum matching* problem.

The challenge here lies in the fact that it is extremely time-consuming to run repair algorithm between every die pairs and hence it is impractical to obtain the exact edge information as modeled above. We hence need more efficient method to address the die matching problem, as formulated in the following.

Problem: Given a number of memory dies with the following information:

- The number of memory blocks in each die;
- The fault bitmap of each memory block in every die obtained from pre-bond testing;
- The number of spare rows/columns of each memory block;

Our objective is to match the given dies in a *pairwise* manner to maximize the total number of reparable 3D-stacked memory circuits. It is important to note that even though a 3D-stacked memory can contain more than two layers, we consider pairwise matching only in this work. There are two reasons behind: (i). selectively matching more than two dies is a much more complicated problem since the possible combinations grow rapidly; (ii). generally speaking, the die yield cannot be a small value (otherwise the product is not profitable) and a good pairwise matching algorithm is able to recover most self-irreparable dies. By simply combining the matched memory pairs to form 3D-stacked memory circuits with more layers, we can maintain a high memory yield.

5.2 Direct Matching

As discussed earlier, we cannot afford the computational complexity to run repair algorithm for every possible pair of memory dies to obtain *exactly* whether two dies matched together can form a reparable stacked circuit (referred to as *matching condition*). If, however, we can *estimate* the *matching conditions* with sufficient accuracy *efficiently*, we should be able to achieve good matching results. Motivated by the above, we develop two kinds of matching conditions to solve our die matching problem. Before describing them in detail, we first present our die matching algorithm as follows.

5.2.1 Die Matching Algorithm

Fig. 5 presents the pseudocode of our algorithm. We first construct a graph with each memory die as a vertex (Line 1), and check every pair of memory dies whether they are considered to be reparable according to our matching condition (Line 4). After checking all pairs and adding the corresponding edges, a undirect graph is constructed (Lines 1-6). We then use the classic *'Blossom' maximum matching algorithm* [11] to get the matched die pairs (Line 7). Finally, to verify whether every pair is reparable and get the final yield (Lines 8-10), we use a two-step memory repair algorithm: (i) We first utilize the efficient ESP algorithm [5] for repair; (ii) If a solution cannot be found, we resort to a branch-and-bound method for another try and abort to repair if a solution still cannot be found under a runtime limit.

	Input: Set of N Memory Die $C = c_1, .., c_N$, Match_Condition
	Output: Set of X Stacking Memory Die $S = s_1, .., s_X$
	Match_Repair(Input,Output,Match_Condistion)
1	Construct a undirect graph $G = \{V, E\}$, $V = C$ $E = \oslash$
2	**for** $i = 1$ to N
3	**for** $j = i+1$ to N
4	Check **Match_Condition** between c_i and c_j
5	If c_i and c_j are considered as reparable
6	$E = E \cup \{c_i, c_j\}$
7	Apply 'Blossom' Maximum-Matching-Algorithm to G, get pair of dies $S = \{(c_i, c_j), ..., (c_i^x, c_j^x)\}$
8	**for** $k = 1$ to x
9	check whether $S_k = (c_i^k, c_j^k)$ is reparable
10	If not, remove S_k from S, x = x - 1
11	Output S

Figure 5: Die Matching Algorithm

5.2.2 Reparability Condition

Let us first consider *reparability condition* which guarantees that any pair of memory dies passing through this condition must be reparable. Apparently, matching self-reparable dies together is one type of reparability condition, as shown in Fig.6(a). However, such matching strategy is too conservative, far from the optimal matching solution[1]. We therefore propose to extend the *ESP* algorithm presented in [5] for building more effective reparability condition efficiently.

Similar to [5], for a memory block i with R_i spare rows and C_i spare columns, we classify the faulty bits into three types: (i). F_i^r bits that are suitable for row repair; (ii). F_i^c bits that are suitable for column repair; and (iii). F_i^o orthogonal bits that can be repaired by either spare rows or spare columns. Then, we use the following formula to determine whether two blocks a and b are reparable after stacking:

$$R_l = (R_a + R_b - F_i^r - F_j^r) \geq 0 \quad (1)$$

[1]The "optimal matched dies" shown in Fig.6 refers to the obtained dies in the case that we can run final repair algorithm between every possible memory die pair to determine whether they are reparable.

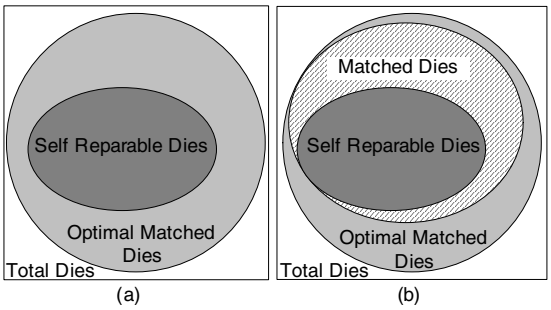

Figure 6: Matching according to Reparability Condition.

$$C_l = (C_i + C_j - F_i^c - F_j^c) \geq 0 \quad (2)$$

$$R_l + C_l \geq F_i^o + F_j^o \quad (3)$$

R_l/C_l are the number of left spare rows/columns after repairing the corresponding faulty cells that require dedicated row/column repair. If their sum can be used to repair the remaining orthogonal faulty bits, we can guarantee that bonding the two memory blocks together is reparable. By checking all pairs of memory blocks between the two memory dies, the above condition is used as our new reparability condition, which can provide yield enhancement when compared to matching self-reparable dies only, as shown in Fig.6(b).

5.2.3 *Irreparability Condition*

The above reparability condition guarantees that all found die pairs are reparable. Such stringent requirement prevents us from finding those reparable pairs that cannot pass the previous reparability condition checking and inherently limits the achievable maximum yield. In this subsection, we consider another type of matching conditions based on *irreparability checking*. By only eliminating those pairs that are guaranteed to be irreparable, more possible die pairs can be found but we cannot guarantee they are reparable. We obtain the irreparability condition by analyzing the property of bipartite graph constructed from the fault bitmap.

Given a bipartite graph $G = (V, E)$, where vertices are partitioned into two sets (X and Y, $x_i \subseteq X$, $y_i \subseteq Y$) and each edge $(x_i, y_i,) \subseteq E$, we have the theorem that the minimum number of vertices that cover all the edges is equal to the number of edges in any *maximum bipartite matching* of the graph [9]. For a memory block i with R_i spare rows and C_i spare columns, it is easy to deduct from the above that, the memory block is not reparable if $R_i + C_i$ is smaller than the number of vertices required for maximum bipartite matching (can be obtained efficiently as in [9]). Now let us consider the matching condition for two memory blocks a and b, we have the following lemma.

Lemma 1: Given two memory blocks with spare rows R_a, R_b and spare columns C_a, C_b. The mapped bipartite graph from these two blocks is $G_a = (V_a, E_a), G_b = (V_b, E_b)$. The maximum bipartite matching of G_a, G_b are M_a, M_b. Then, only if

$$|M_a| + |M_b| \leq R_a + R_b + C_a + C_b \quad (4)$$

the stacked memory is possibly to become reparable.

The die matching algorithm is then conducted on the bipartite graph constructed according to the above irreparability condition. With this strategy, we are able to find more die pairs that are possibly reparable, but this is not guaranteed. For the two examples shown in Fig. 7, both require at least four spares (either spare row or spare column) to be possibly reparable. Even though both are deemed as possibly reparable according to the above lemma, if in total we have 2 spare rows and 2 columns with redundancy sharing, the faults in (a) are reparable (rows 1,2 and columns 6,9), but the faults in (b) cannot be repaired (a possible repair solution is when we have 4 spare columns). Fig. 8 presents the matching effect according to irreparability condition. As can be observed, within the matched dies, only part of them are reparable.

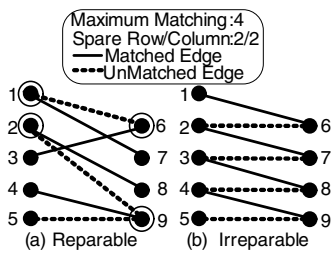

Figure 7: Maximum Matching and Reparability.

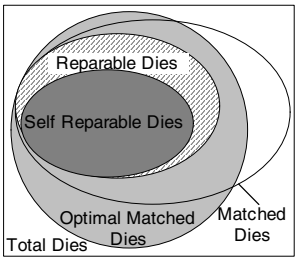

Figure 8: Matching according to Irreparability Condition.

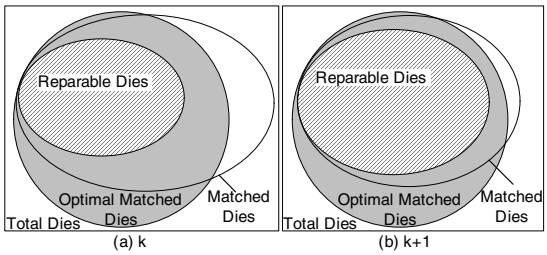

Figure 9: Iterative Matching with Changing Conditions.

5.3 Iterative Matching

Directly matching memory dies according to the previous reparability condition and irreparability condition all have their limitations. In this subsection, we consider to match dies iteratively. That is, we conduct die matching multiple times, and in each iteration, we keep those good pairs while redo the matching for the left dies by changing the matching condition.

We start from the irreparability condition [2] as discussed earlier, since such strategy gives us more possible die pairs and our die matching algorithm will try to match them by aggressively making use of the redundant resources. Then, we keep those pairs that are reparable, and for those found pairs that are not reparable, we conduct matching again with tightening irreparability condition as follows:

$$|M_i| + |M_j| + k \leq R_i + R_j + C_i + C_j \quad (5)$$

, with gradual increase of the value k (initialized as 0).

As shown in Fig. 9, every time we tighten the irreparability condition, more reparable pairs are found because the existence of more spare rows/columns between them. At the same time, the number of found pairs decreases, and the above procedure terminates until we cannot find any pairs that can be matched.

6. EXPERIMENTAL RESULTS

6.1 Experiment Setup

We consider a total of 1000 1Gb memory dies to be formed as 2-layer memory circuits, and hence we can obtain a maximum of 500 functional stacked memories when the yield is 100%. Each memory die contains 4×4 memory blocks, and each memory block is with the size of $8k \times 8k$ ($Row \times Column$) bit-cells.

For fault injection, we consider two distributions to obtain the number of faults in each die: (i). Poisson distribution with $\lambda = 2.130$ [6]; (ii). Polya-Eggenberger distribution with $\lambda = 2.130$ [5]. For Polya-Eggenberger distribution, we also tune another parameter α with $\alpha = 2.382$ [13] and $\alpha = 0.6232$ [14], representing the case with clustered faults and that with evenly distributed faults, respectively. We assume that all the spare rows/columns can be borrowed between neighboring vertical memory blocks, and we inject random TSV faults with faulty rate as 0.1%. Experiments are conducted on two cases with different percentage of six kinds of faults (see the following table).

Fault	Single Cell	Double Cell	Single Row	Single Column	Double Row	Double Column
Case 1	40%	4%	20%	20%	8%	8%
Case 2	70%	4%	8%	8%	5%	5%

Table 1: Experimental Parameters for Two Cases.

6.2 Results and Discussion

In our experiments, we compare four matching strategies: (i). matching self-reparable dies only; (ii). matching according to reparability condition; (iii). matching according to irreparability condition; and (iv). iterative matching. Tables 2 present our experimental results with gradual increase of spare rows/columns.

[2] Starting from reparability condition does not give us any benefit because all the matched pairs are reparable already.

Table 2: Experimental Results

Spare	Self Repair		Reparability Condition		Irreparability Condition			Iterative Approach	
$SR \times SC$	N_{repair}	Y_{self}	N_{repair}	ΔY_r	N_{match}	N_{repair}	ΔY_{ir}	N_{repair}	ΔY_{it}
Poisson Distribution									
6×6	203	40.6%	341	27.6%	386	281	15.6%	352	29.8%
8×8	268	53.6%	415	29.4%	475	367	19.8%	446	**35.6%**
10×10	358	71.6%	481	24.6%	496	411	10.6%	**482**	24.8%
12×12	388	77.6%	485	19.4%	498	448	12.0%	489	20.2%
14×14	428	85.6%	496	13.6%	500	477	9.8%	496	13.6%
16×16	457	91.4%	498	8.2%	500	490	6.6%	498	8.2%
18×18	**471**	94.2%	499	5.6%	500	498	5.4%	499	5.6%
Average				18.34%			11.4%		19.7%
Polya-Eggenberger Distribution $\alpha = 0.6232$									
6×6	307	61.4%	412	21.0%	445	404	19.4%	428	**24.2%**
8×8	367	73.4%	436	13.8%	483	407	8.0%	460	18.6%
10×10	410	82.0%	457	9.4%	494	435	5.0%	**480**	14.0%
12×12	434	86.8%	477	8.6%	497	454	4.0%	485	10.2%
14×14	453	90.6%	490	7.4%	499	469	3.2%	490	7.4%
16×16	463	92.6%	495	6.4%	500	484	4.2%	496	6.6%
18×18	**471**	94.2%	497	5.2%	500	489	3.6%	497	5.2%
Average				10.26%			6.77%		12.31%
Polya-Eggenberger Distribution $\alpha = 2.38$									
6×6	111	22.2%	202	18.2%	215	169	11.6%	205	18.8%
8×8	152	30.4%	278	25.2%	318	181	5.8%	286	26.8%
10×10	187	37.4%	340	30.6%	396	179	-1.6%	**350**	**32.6%**
12×12	219	43.8%	354	27.0%	414	254	7.0%	377	31.6%
14×14	257	51.4%	393	27.2%	447	300	8.6%	406	29.8%
16×16	292	58.4%	421	25.8%	457	353	12.2%	430	27.6%
18×18	**319**	63.8%	443	24.8%	477	374	11.0%	450	26.2%
Average				25.54%			7.8%		27.62 %

ΔY_r: Yield improvement according to reparability condition over Y_{self}.
ΔY_{it}: Yield improvement with iterative matching over Y_{self}.

(a) Case1

Spare	Self Repair		Reparability Condition		Irreparability Condition			Iterative Approach	
$SR \times SC$	N_{repair}	Y_{self}	N_{repair}	ΔY_r	N_{match}	N_{repair}	ΔY_{ir}	N_{repair}	ΔY_{it}
Poisson Distribution									
6×6	291	58.2%	446	31.0%	455	445	30.8%	452	**32.2%**
8×8	376	75.2%	493	23.4%	495	485	21.8%	493	23.4%
10×10	443	88.6%	500	11.4%	500	494	10.2%	**498**	11.0%
12×12	462	92.4%	500	7.6%	500	499	7.4%	499	7.4%
14×14	483	96.6%	500	3.4%	500	500	3.4%	500	3.4%
16×16	494	98.8%	500	1.2%	500	500	1.2%	500	1.2%
18×18	**498**	99.6%	500	0.4%	500	500	0.4%	500	0.4%
Average				11.2%			10.74%		11.29%
Polya-Eggenberger Distribution $\alpha = 0.6232$									
6×6	382	76.4%	468	17.2%	477	469	17.4%	474	**18.4%**
8×8	436	87.2%	486	10.0%	495	479	8.6%	490	11.0%
10×10	468	93.6%	492	4.8%	499	488	4.0%	**495**	5.0%
12×12	476	95.2%	496	4.0%	500	495	3.8%	498	5.0%
14×14	484	96.8%	498	2.8%	500	497	2.6%	498	2.8%
16×16	491	98.2%	499	1.6%	500	498	1.4%	499	1.8%
18×18	**494**	98.8%	499	1.0%	500	499	1.0%	499	1.2%
Average				5.91%			5.54%		6.2%
Polya-Eggenberger Distribution $\alpha = 2.38$									
6×6	177	35.4%	294	23.4%	305	294	23.4%	302	25.0%
8×8	228	45.6%	371	28.6%	389	354	25.2%	378	**30.0%**
10×10	298	59.6%	416	23.6%	439	394	19.2%	**429**	26.2%
12×12	313	62.6%	438	25.0%	452	423	22.0%	444	26.2%
14×14	354	70.8%	456	20.4%	472	441	17.4%	461	21.4%
16×16	382	76.4%	472	18.0%	481	468	17.2%	476	18.8%
18×18	**408**	81.6%	479	14.2%	489	469	12.2%	482	14.8%
Average				21.89%			19.51%		23.2%

ΔY_{ir}: Yield improvement according to irreparability condition over Y_{self}.

(b) Case 2

From this table, we can observe that, with redundancy sharing, all the three proposed matching strategies significantly increase memory yield when compared to the case that we bond self-reparable dies only (up to 35.6% yield enhancement). Generally speaking, the amount of yield improvement decreases with the increase of redundancy, while the number of self-reparable dies grows rapidly. From another perspective, we can observe that for all the cases, iterative matching with 10×10 spares outperforms bonding self-reparable dies only with 18×18 spares. In other words, to achieve the same memory yield, much fewer spare rows/columns are needed with redundancy sharing, which justifies the use of some TSVs to facilitate this repair strategy.

In most cases, iterative matching results in the highest yield, followed by matching according to reparability condition while matching according to irreparability condition is inferior to the other two strategies. This is because, even though the number of matched dies according to irreparability condition is usually quite high (see Column N_{match}), a non-trivial portion of them might be irreparable, especially when the redundant resources are not sufficient. There are also few cases that iterative matching results in less reparable memory circuits when compared to matching according to reparability condition (e.g., Poisson Distribution in Table 2(b) with 10×10 spares). This is because, when the first iteration matching according to irreparability condition has returned very high yield (494 out of 500 die pairs are reparable in this case), there are few flexibility for us to match more reparable pairs for the left ones.

From Table 2, we can also see that the yield in Case 2 is always higher than that in Case 1. This is because, there are much more single cell faults in Case 2 and they can be easily repaired with either a spare row or a spare column, and hence the repair algorithm has higher flexibility to fulfill such needs. For polya-eggenberger fault distribution, the yield values with $\alpha = 0.6232$, is much larger than that with $\alpha = 2.38$. This is also expected because, there are more fault clusters when α is large, and hence such memory dies are more probable to have irreparable memory blocks even with redundancy sharing.

7. CONCLUSION

In this paper, we propose to conduct redundancy sharing across neighboring dies for yield enhancement of 3D-stacked memory circuits. We first develop a repair strategy that enables redundancy sharing for any given pair. Next, we present novel solutions that selectively match memory dies together for yield maximization. Experimental results show that the proposed technique is able to greatly enhance memory yield or requires much less redundant resources to achieve similar yield.

8. ACKNOWLEDGEMENTS

This work was supported in part by Hong Kong SAR Research Grants Council (RGC) under the General Research Fund CUHK417807, and CUHK418708, in part by National Science Foundation of China (NSFC) under grant No. 60876029, and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

9. REFERENCES

[1] S. Bahl. A Sharable Built-in Self-repair for Semiconductor Memories with 2-D Redundancy Scheme. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2007.

[2] K. Chakraborty and P. Mazumder. *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*. Prentice Hall PTR, 2002.

[3] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon. Demystifying 3D ICs: The Pros and Cons of Going Vertical. *IEEE Design & Test of Computers*, 22(6):498–510, 2005.

[4] H. Fernau and R. Niedermeier. An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover. *Journal of Algorithms*, 38(2):374–410, 2001.

[5] C. Huang, C. Wu, J. Li, and C. Wu. Built-in Redundancy Analysis for Memory Yield Improvement. *IEEE Transactions on Reliability*, 52(4):386–399, 2003.

[6] R. Huang, J. Li, J. Yeh, and C. Wu. A Simulator for Evaluating Redundancy Analysis Algorithms of Repairable Embedded Memories. In *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, pp. 68–73.

[7] U. Kang, et al. 8Gb 3D DDR3 DRAM using Through-Silicon-Via Technology. In *Proc. ISSCC,* pp. 130–131, 2009.

[8] L. Jiang, L. Huang and Q. Xu Test Architecture Design and Optimization for Three-Dimensional SoCs. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)* pp. 220-225, 2009

[9] S. Kuo and W. Fuchs. Efficient Spare Allocation in Reconfigurable Arrays. In *Proceedings of the 23rd ACM/IEEE conference on Design automation*, pp. 385–390. IEEE Press Piscataway, NJ, USA, 1986.

[10] C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari. Bridging the Processor-Memory Performance Gap with 3D IC Technology. *IEEE Design & Test of Computers*, 22(6):556–564, 2005.

[11] S. Micali, V. Vazirani, K. Fujita, Y. Nishimura, and K. Anami. An $O(V^{1/2}E)$ Algorithm for Finding Maximum Matching in General Graphs. *21st Annual Symposium on Foundations of Computer Science*, pp. 17-27, 1980.

[12] S. Reda, G. Smith, and L. Smith. Maximizing the Functional Yield of Wafer-to-Wafer 3D Integration. *IEEE Transactions on VLSI Systems, to appear*, 2009.

[13] C. Stapper, A. McLaren, and M. Dreckmann. Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product. *IBM J. Res. Develop*, 24(3):398–409, 1980.

[14] C. Wey and F. Lombardi. On the Repair of Redundant RAM's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(2):222–231, 1987.

[15] T. Yamagata, H. Sato, K. Fujita, Y. Nishimura, and K. Anami. A Distributed Globally Replaceable Redundancy Scheme for Sub-Half-Micron ULSI Memories and Beyond. *IEEE Journal of Solid-State Circuits*, 31(2):195–201, 1996.