OptiSIMPLER: Optimal Single-Row Technology Mapping for MAGIC using SAT

Debjyoti Bhattacharjee
School of Computer Science and Engineering, Nanyang Technological University, Singapore

***Abstract*—This work presents an optimal approach for implementing single-instruction multiple-data (SIMD) like in-memory operations. This work demonstrates computing of an arbitrary logic function onto a single row of a memristive array, while allowing device reuse. This mapping can be replicated across multiple rows of the array for SIMD operations, harnessing the parallelism offered by the memristor-aided logic NOR gates. The OptiSIMPLER framework has been implemented in Python. The framework performs mapping with either minimum number of devices or minimum number of cycles, or both. The proposed framework supports logic functions mapped to NOT and NOR gates with arbitrary number of inputs (> 2).**

I. Preliminaries and Problem Formulation

We present a method for determining the minimum number of devices required for realizing a mapped netlist using MAGIC operations. We consider the following assumptions. There is a single row of memristors — this allows only a single MAGIC operation per cycle and the position of the operands in the row does not impact the overall delay of mapping. Each primary input present in the netlist, is already allocated a device, prior to the start of computation. A gate in the netlist can be computed exactly once. A device that has the result of a computation that is not required any longer, can be set. Once an output has been computed on a memristor, the memristor cannot be set.

The key difference from the constraints used in SIMPLE [1] comes from the fact that the current work does not consider the geometry of the crossbar in the formulation. This allows considerable reduction in the number of constraints, since mapping a node to any device is same as mapping to any other device. Furthermore, the constraints present in SIMPLE that have to be solved to permit parallel execution of nodes are not present in the current formulation.

II. SAT formulation for optimal single-row execution of MAGIC operations

In this section, we present OptiSIMPLER — a procedure that determines a feasible sequence of operations to computed a given mapped netlist using a given number of available memristors using SAT constraints.

A. Inputs

The inputs provided to OptiSIMPLER are summarily presented in Table I.

TABLE I: Input parameters to OptiSIMPLER

Parameter	Description
G	A graph $G = \langle V, E \rangle$ is created from a mapped NOR/NOT netlist. Each gate g_i is represented by a vertex i in the graph. An edge $i \rightarrow j \in G$, if the output of gate g_i is an input to gate g_j.
PO	A subset of vertices V that represent the gates that drive the primary outputs.
D	D is the number of devices available for mapping, $D > 1$.
S	S is a positive integer that determines the number of steps that are available to the SAT solver for mapping. [Refer to example for details.]

B. Variables used by SAT

Variable	Type	Description
a_i^s	Binary	1 indicates that vertex i is assigned to a device in step s. 0 indicates that no device has been allocated to vertex i in step s.
f_i^s	Binary	1 indicates that vertex i was assigned a device in some step s', such that $s' \leq s$.

C. Clauses for SAT formulation

1) No vertex has a device allocated at the beginning.

$$a_i^0 = 0, \forall i \in V \tag{1}$$

2) Each output vertex must have a device allocated at the end.

$$a_i^S = 0, \forall g_i \in PO \tag{2}$$

3) A vertex can be allocated only if all the predecessors of the node are currently allocated or if it is already allocated.

$$(a_{p1}^{s-1} \wedge a_{p2}^{s-1} \wedge \ldots \wedge a_{ik}^{s-1}) \vee a_i^{s-1} \implies a_i^s \tag{3}$$

where $pi \in pred(i), 1 \leq s \leq S$.

4) In each step, the number of devices allocated cannot be more than number of available devices D.

$$\textsc{AtMost}(a_1^s, a_2^s, \ldots, a_{|V|}^s, D), \forall 1 \leq s \leq S \tag{4}$$

5) A vertex can be allocated only once. This is also known as *one-shot* computation. $\forall i \in V$ and $1 \le s \le S$:-

$$f_i^s = f_i^{s-1} \vee (a_i^s \wedge (\neg a_i^{s-1})) \quad (5)$$
$$f_i^s \implies \neg a_i^s \quad (6)$$

D. MAGIC operations from SAT solver assignment

The output of the SAT solver is a feasible assignment of the variables, such that all the constraints are True. We can construct a valid sequence of MAGIC operations from the assignment of the variables a_i^s using the Algorithm 1.

Algorithm 1: Determining sequence of MAGIC operations from SAT solver assignment A

```
1  Procedure MAGICOpGen(G, A, D, S)
2      Ops = List();
3      verToDev = Map();
4      freeDev = Stack();
5      for $d = 1; d \le D; d++$ do
6          freeDev.push(d);
7      for $s = 1; s \le S; s++$ do
8          for $i = 1; i \le |V|; i++$ do
9              if $\neg a_i^s \&\& a_i^{s-1}$ then
10                 dev = verToDev[i];
11                 freeDev.Push(dev);
12                 Ops.add(dev,Set);
13         for $i = 1; i \le |V|; i++$ do
14             if $a_i^s \&\& \neg a_i^{s-1}$ then
15                 dev = freeDev.Pop();
16                 verToDev[i] = dev ;
17                 Ops.add(dev,$g_i$);
18     return Ops ;
```

III. Determining minimum number of devices

The SAT formulation can be used to determine the minimum number of devices required for single-row computation of an arbitrary mapped netlist. The algorithm 2 presents the procedure for the same. The algorithm uses $\log_2 |V|$ calls to the OptiSIMPLER to determine the minimum number of devices. It should be noted that the amount of time that the SAT solver takes to complete execution varies for different values of available devices D, while keeping the other inputs identical. Therefore, the overall time for completing the minimum search cannot be estimated directly from one call to OptiSIMPLER.

IV. Demonstrative Example

Fig. 1 shows a representative DAG. Nodes 0—7 without any fan-in are the primary inputs while node 8 is the primary output. For the constraint formulation, we set the number of steps S to 10 for this example.

As stated in Algorithm 2, the procedure uses a binary search to find the minimum number of devices. Table II shows the result of each SAT solver invocation, along with the corresponding delay achieved. Note that for each call to the SAT solver, we use the same number of steps ($2|V|$) available for mapping. The minimum number of devices that gives a feasbile mapping for the DAG is equal to 5.

The solution obtained using 5 devices is shown in Table III. We should note that each time step s does not correspond directly to the cycles of executing MAGIC operations. If there

Algorithm 2: Determining minimum number of devices using SAT

```
1  Procedure FindMinDev(G, PO)
2      D = |V| ;
3      S = 2D ;
4      feasible,feasibleAlloc = OptiSIMPLER(G,
         PO, D, S);
5      feasibleDev = D;
6      high = D;
7      low = 1;
8      while $low \le high$ do
9          mid = $(low + high)/2$;
10         feasible, alloc = OptiSIMPLER(G, PO,
             mid, S);
11         if $feasible == sat$ then
               // The solver found a
                  satisfiable assignment
12             feasibleAlloc = alloc;
13             feasibleDev = mid;
14             high = $mid - 1$;
15         else
16             low = $mid + 1$;
17     return feasibleDev, MAGICOpGen(G,
         feasibleAlloc, feasibleDev, S);
```

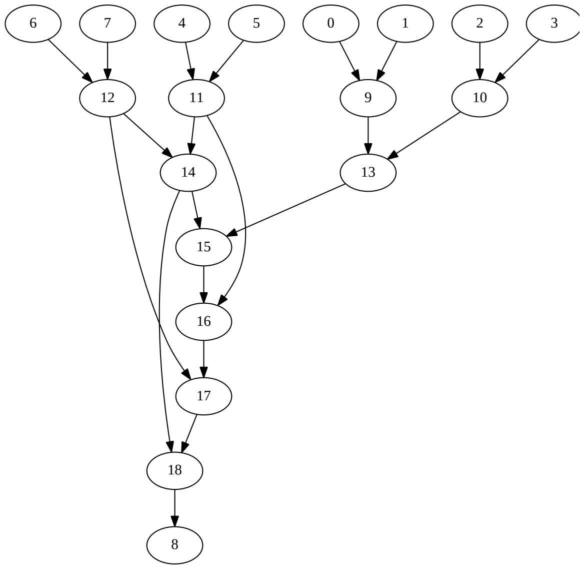

Fig. 1: A representative DAG.

TABLE II: Iterative binary search (over D) to determine mininum number of devices required for mapping.

D	Result	Cycles	MAGIC	SET	
10	sat	13	11	2	
5	sat	16	11	6	
4	unsat	—	—	—	—
3	unsat	—	—	—	—

is more than one operation in a time step, the operations are performed sequentially — the SET operations first, followed by the MAGIC operations. Initially at $s = 0$, none of the devices are allocated a device and hence the all the entries are 0. At step $s = 1$, nodes 11 and 12 are assigned a device and computed. Similarly, 9 and 10 are allocated and computed in $s = 3$. In $s = 4$, 13 is computed. In step $s = 5$, the devices allocated to 9 and 10 are SET, and one of these devices are used for computing 14. Similarly, rest of the assignments can be interpreted. Note that, we consider only one time step in the assignment, if the assignment in consecutive times steps is identical (for e.g., $s = 1$ and $s = 2$). Using Algorithm 1, the sequence of MAGIC operations corresponding to the assignment is presented in Fig. 2.

V. Results

The results of benchmarking are presented in Table IV. The proposed algorithm was implemented in Python, with Z3 as the constraint solver. The experiments were run on Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz, with 30GB RAM, running Ubuntu 12.04. The runtime of each benchmark was limited to roughly 24 hours. All the benchmarks were in the form of a mapped netlist, consisting of NOR and NOT gates only. The *minDev* column indicates the minimum number of devices for which a satisfiable solution (sat) was found while the *unsatDev* indicates the number of devices for which feasible solution could not be found. The results where $minDev \neq unsatDev + 1$, indicate that the solver was terminated before the optimal solution could be derived. The Cycles columns indicate the delay of mapping, consisting of MAGIC and *Set* operations. $\Delta\%$ indicates the percentage reduction in device usage, compared to the trivial allocation (equal to number of nodes), considering one device allocated per node.

TABLE III: Solution obtained from the SAT solver for $D = 5$ — Assignment of device to node.

s	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}	n_{14}	n_{15}	n_{16}	n_{17}	n_{18}
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0
2	0	0	0	1	1	0	0	0	0	0	0
3	0	1	1	1	1	0	0	0	0	0	0
4	0	1	1	1	1	1	0	0	0	0	0
5	0	0	0	1	1	1	1	0	0	0	0
6	0	0	0	1	1	1	1	1	0	0	0
7	0	0	0	1	1	0	1	1	1	0	0
8	0	0	0	0	1	0	1	1	1	1	0
9	0	0	0	0	1	0	1	1	0	1	1
10	1	0	0	0	1	0	0	1	0	0	1

Cycle	Op.	Node	#Dev
1	MAGIC	11	1
2	MAGIC	12	2
3	MAGIC	9	3
4	MAGIC	10	4
5	MAGIC	13	5
6	SET	9	3
6	SET	10	4
7	MAGIC	14	4
8	MAGIC	15	3
9	SET	13	5
10	MAGIC	16	5
11	SET	11	1
12	MAGIC	17	1
13	SET	16	5
14	MAGIC	18	5
15	SET	17	1
16	MAGIC	8	1

Fig. 2: Result of optimal mapping using 5 devices. The mapping requires 16 cycles — 11 MAGIC and 6 SET operations.

TABLE IV: Benchmarking results for SAT-based single row mapping.

Benchmark	Nodes	min	unsat	MAGIC	Set	Cycles	$\Delta\%$	$\alpha\%$
x2	78	19	NA	68	49	117	75.64	50.00
misex	86	10	7	78	68	146	88.37	69.77
parity	92	11	NA	76	65	141	88.04	53.26
5xp1	119	30	NA	112	82	194	74.79	63.03
clip	161	40	NA	152	112	264	75.16	63.98
b9	188	21	20	147	126	273	88.83	45.21
apex2	374	93	NA	336	243	579	75.13	54.81
x4	547	102	NA	453	351	804	81.35	46.98
aes	365	45	NA	357	312	669	87.67	83.29

$\alpha\%$ is the percentage overhead in terms of number cycles, compared to the delay achievable using trivial allocation.

VI. Conclusion

The report presented an algorithm to obtain the minimum number of devices required for mapping arbitrary netlist using a single row of memristors realizing MAGIC operations. Using a SAT solver, an optimal solution can be obtained for an arbitrary number of available devices for mapping. Benchmarking results indicate that considerable reduction in device usage can be obtained (upto 88%) by permitting device reuse, but at the cost increased delay. Further reduction in device usage might be achieved by permitting computation of nodes, more than once.

References

[1] R. B. Hur, N. Wald, N. Talati, and S. Kvatinsky, "Simple magic: Synthesis and in-memory mapping of logic execution for memristor-aided logic," in *Proceedings of the 36th International Conference on Computer-Aided Design*, pp. 225–232, IEEE Press, 2017.