Parallel Layer-Aware Partitioning for 3D Designs

Yi-Hang Chen
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan
carlok@adar.ee.nctu.edu.tw

Yi-Ting Chen
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan
wallet0530.ee99g@nctu.edu.tw

Juinn-Dar Huang
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan
jdhuang@mail.nctu.edu.tw

Abstract –As compared to two-dimensional (2D) ICs, 3D integration is a breakthrough technology of growing importance that has the potential to offer significant performance and functional benefits. This emerging technology allows stacking multiple layers of dies and resolves the vertical connection issue by through-silicon vias (TSVs). However, though a TSV is considered a good solution for vertical connection, it also occupies significant silicon estate and incurs reliability problem. Because of these challenges, minimizing the number of TSVs becomes an important design issue. Therefore, in this paper, we propose a two-phased parallel layer-aware partitioning algorithm for TSV minimization in 3D structures. In the first phase, we employ OpenMP to parallelize the 2-way min-cut partitioning and get the initial solution. In the second phase, we further improve the result using a parallel simulated annealing approach on GPU. Experimental results show that the proposed algorithm can reduce the number of TSVs by about 39% as compared to several existing methods.

I Introduction

As fabrication process exploits even deeper submicron technology, device feature size and interconnect width are continuously scaling down. Hence, there are numerous tough challenges raised, such as signal integrity, power integrity, heat dissipation, leakage power, clock distribution, and yield issues [1]. Furthermore, it also causes global interconnect delay become a dominating factor of system performance and throughput. However, traditional scaling is no longer the best way for advancing manufacturing process technology due to the physical limitations. Hence 3D integrated circuit technologies have been emerging in recent years [2]–[5]. Through-silicon via (TSV) is one of the most promising methods to accomplish vertical interconnects between different layers among several state-of-the-art 3D integration technologies [4]. Nevertheless, compared with a typical 2D design, though a TSV-based 3D design can generally reduce the global interconnect delay, currently available TSV fabrication processes still suffer from relatively low yield as well as large area overhead [6]. Consequently, the issue of TSV minimization must be properly addressed in a design flow as stepping into the 3D IC era.

In past few years, several previous works have already been proposed to tackle the problem of 3D partitioning for TSV minimization. One solution is to solve the problem using integer linear programming (ILP) [7]. However, it can only solve small problems since its runtime grows exponentially as problem size increases. In [8][9], each of them develops a modified FM-based [10] partitioning method to obtain the resultant layer assignment. However, all these methods only focus on minimizing the total amount of TSVs or die area, and do nothing about evenly distributing TSVs among layers. Meanwhile, the authors in [11][12] alternatively use a two-step approach – first applying the well-known partitioning algorithm *hMetis* [13] to divide a design into a set of layer-unaware partitions, and then associating each partition with a layer (i.e., layer assignment) – to accomplish 3D design partitioning. After that, Huang *et al.* present an iterative layer-aware partitioning algorithm *iLap* targeting TSV minimization for 3D structures [14]. In this paper, we also utilize *hMetis* as the partitioning engine since it can produce good initial results for further refinement. Chen *et al.* propose a parallel partitioning algorithm based on a force-directed scheme on GPGPU [15]. However, the results are not as good as the ones obtained by *iLap*.

In this paper, we present a two-phased parallel layer-aware partitioning algorithm for 3D structures. At first, we propose a two-step deterministic method for finding a feasible solution in parallel. Then we further refine the solution by exposing the massive parallelism provided by the general-purpose GPU architecture. Though the proposed method utilizes *iLap* as the kernel of its partitioning engine in the first phase, the experiment results demonstrate that our method can apparently do better TSV minimization than *iLap* for various number of layers, and also can reduce the runtime by about 70%.

The rest of this paper is organized as follows. Section II details the proposed two-phase parallel layer-aware partitioning algorithm. Experimental results and analyses are reported in Section III. Finally, the concluding remarks are given in Section IV.

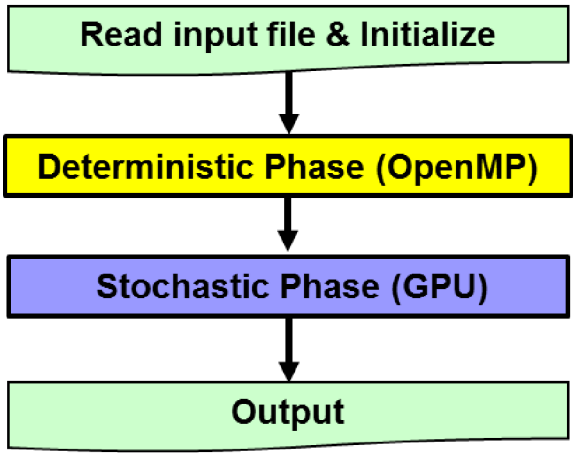

Figure 1. Overall flow of the proposed method.

II. Proposed Algorithm

In this section, we present our two-phase parallel layer-aware partitioning algorithm. Figure 1 shows the overall flow of our proposed method. Each phase is described in detail below.

A. *Problem Formulation*

A design is modeled as a hypergraph $G = (V, E)$, where V is a set of vertices including a set of functional cells C and a set of I/O pads I (i.e., $V = C \cup I$, $C \cap I = \emptyset$); and E is a set of hyperedges. For each vertex $v \in V$, $area(v)$ denotes the area cost of v. A k-layer disjoint partition set of G with all the I/O terminals residing in the bottom-most layer is represented as $L = \{L_0=I, L_1, L_2 \ldots L_k\}$, where L_i is the partition assigned to the i-th layer and is a subset of C. For each layer L_i, A_{Li} denotes the summation of area(v), $\forall v \in L_i$. And A_{avg} is used to indicate the average of the total area in the k layers, which is calculated by:

$$A_{avg} = \sum_{v \in V} area(v)/k.$$

We further define the area constraint as:

$$A_{avg} \times (1 - r) \leq A_{Li} \leq A_{avg} \times (1 + r)$$

with a balanced ratio r, $0 < r < 1$, for balancing the area of layers.

In this work, we model the 3D partitioning problem as a layer-aware multi-way partitioning problem. Given a target 3D structure consisting of k layers stacking vertically, a design G, and the I/O pad constraint, our proposed algorithm partitions G into k sub-designs in parallel and each sub-design is explicitly associated with a vertical layer so that the total number of TSVs is minimized.

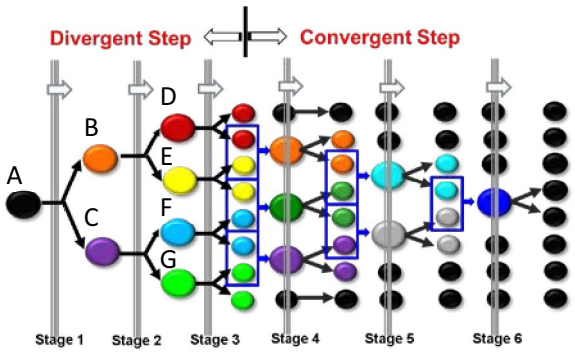

Figure 2. Detailed procedures of deterministic phase (k=8).

B. *Deterministic Phase*

Deterministic phase can be divided into two steps, divergent step and convergent step. In the divergent step, 2-way min-cut partitioner is applied in each stage in parallel until the design has been partitioned into k parts. However, the process is sometimes arbitrary due to the flexibility of vertices which have been limited in the early stages. As a result, we proposed the convergent step as a fine tuning step. The step works in a merge-and-split fashion, it iteratively merges adjacent layers that are belong to different parts in the last stage and then partition again until no more layers can be merged. Below we use a simple 8-layer 3D partitioning example shown in Figure 2 to illustrate the two steps in more detail.

Firstly, 2-way min-cut partitioner is applied on the whole design to get two partitions, part B and part C. Next, we further apply 2-way min-cut partitioner on part B and C in parallel to get four partitions, part D~G, as Figure 2 depicts. At last, 2-way min-cut partitioner has been performed in parallel on part D~G to get final eight partitions and finish divergent step. In the following convergent step, we merge adjacent layers (blue rectangles shown in Figure 2) that are belong to different parts in the last stage and apply 2-way min-cut partitioner on it in parallel again. The step repeatedly executed until no more layers can be merged.

C. *Stochastic Phase*

In this phase, we try to utilize the computing power of state-of-the-art GPUs to further reduce the number of TSVs. We propose a parallel simulated annealing approach to explore the solution space and search for better solutions. Figure 3 shows the overall flow of this phase. At the beginning, the initial solution obtained from the first phase is used as start point of this phase. Secondly, we use w threads to parallel the simulated annealing procedure. There are two possible perturbations in our algorithm, including swap and move. L perturbations have been probabilistically evaluated for each thread within an iteration step. At the end of iterations, the host (CPU) updates the temperature of system

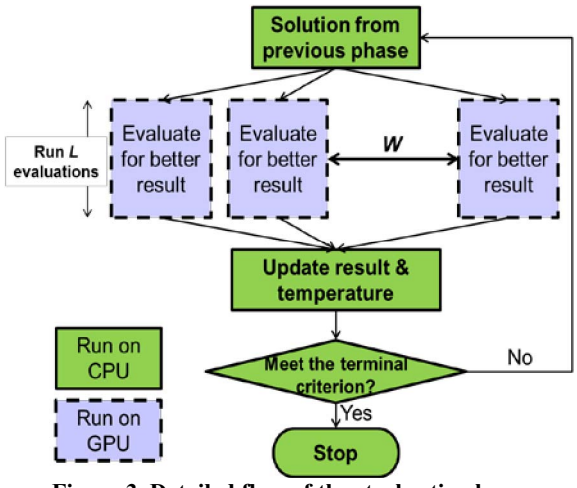

Figure 3. Detailed flow of the stochastic phase.

Table 1. Total number of TSVs with *k*=4.

4 layers	Total TSVs				Normalized to hMetis		
Design	Our	iLap	hMetis	EX-hMetis	Our	iLap	EX-hMetis
Tseng	290	307	363	345	0.80	0.85	0.95
Diffeq	234	242	321	280	0.73	0.75	0.87
Des	446	445	851	824	0.52	0.52	0.97
Bigkey	616	612	666	652	0.93	0.92	0.98
Frisc	616	657	713	687	0.86	0.92	0.96
elliptic	565	589	681	640	0.83	0.87	0.94
pdc	908	1006	1088	1027	0.83	0.92	0.94
fft128	1283	1306	1504	1487	0.85	0.87	0.99
s38417	231	245	351	320	0.86	0.70	0.91
s38584.1	290	392	683	554	0.57	0.57	0.81
clma	483	487	714	487	0.68	0.68	0.68
cfft	228	239	1015	309	0.22	0.24	0.30
aqua	827	895	6893	5085	0.12	0.13	0.74
video	757	737	8719	7129	0.09	0.08	0.82
Average					0.62	0.64	0.85

and evaluates the results to determine the start points for next iteration. The temperature of each iteration decreases with the cooling rate α so the probability of accepting an inferior solution is getting lower, and finally an optimal solution can be obtained. The above steps are repeated until a fixed number of times.

It is worth to mention that SA usually starts with a random generated initial solution. But in our cases, the initial solution from previous phase is far better than the random generated one. Therefore, starting with a high temperature will lead to accept too many uphill moves at the beginning and makes it even harder to reach a better solution. After observing the solution quality of a set of experiment with different temperatures, we recommend to set the value below to 1. However, the improvement is still less than 1% even though we exhaustively try almost all of the combinations of parameters.

III. Experiments

A. Environmental Setup

Our proposed method has been implemented in C++/Linux environment, with an Intel Xeon 2.4GHz CPU. In our second phase, the GPGPU platform is NVidia Tesla m2050. In this section, we demonstrate the effectiveness of our method through a series of comparisons with *iLap* and two hMetis-based methods: 1) *hMetis*: partitions are further layered according to their original sequential tags (i.e., in random order basically); 2) *EX-hMetis*: partitions are best layered through exhaustively examining all possible layer permutations. We evaluate the performance of our method and other three methods over a set of 14 test cases, consisting of 10 cases from the MCNC benchmark set [16], three large cases (*cfft*, *aqua*, and *video*) from Altera [17], and one in-house 128-point FFT design (*fft128*). The number of blocks ranges from 1,047 to 53,491 among all the cases. The parameters of stochastic phase have been set as: W=256, L=100, initial temperature=1 and cooling rate=0.95. We perform ten experiment runs on every test case with different random seeds and take the average as the final result.

Table 2. Total number of TSVs with *k*=8.

8 layers	Total TSVs				Normalized to hMetis		
Design	Our	iLap	hMetis	EX-hMetis	Our	iLap	EX-hMetis
Tseng	677	740	860	789	0.79	0.86	0.92
Diffeq	538	597	760	654	0.71	0.79	0.86
Des	1075	1087	1977	1880	0.54	0.55	0.95
Bigkey	1437	1451	1557	1516	0.92	0.93	0.97
Frisc	1333	1428	1655	1550	0.81	0.86	0.94
elliptic	1205	1321	1587	1416	0.76	0.83	0.89
pdc	1918	2171	2447	2199	0.78	0.89	0.90
fft128	2964	3070	3530	3484	0.84	0.87	0.99
s38417	530	573	837	749	0.63	0.68	0.89
s38584.1	936	1022	1669	1367	0.56	0.61	0.82
clma	1289	1383	1821	1375	0.71	0.76	0.76
cfft	541	542	2474	704	0.22	0.22	0.28
aqua	1828	1978	15864	10795	0.12	0.12	0.68
video	1915	2005	20340	15811	0.09	0.10	0.78
Average					0.61	0.65	0.83

B. Results and Analyses

A set of experiments are conducted with 4 and 8 layers. Table 1 reports the TSV demands as the number of layers is set to 4. *EX-hMetis* always picks the one with the lowest TSV count out of 4! = 24 different layer permutations that can be generated from given a set of 4 partitions generated by *hMetis* and consequently *EX-hMetis* on average attains 15% TSV reduction as compared with *hMetis*. Nevertheless, the proposed method can reduce TSV count by 38% and 27% on average as compared to *hMetis* and *EX-hMetis*, respectively. Moreover, for the largest three test cases (*cfft*, *aqua*, and *video*), our method even outperforms *hMetis* by more than 78%. Unfortunately *hMetis* fails to be a good 3D partitioner due to its layer-unawareness even though it is an excellent multi-way min-cut partitioning algorithm. Even *EX-hMetis*, with exhaustive layer permutations, still cannot beat our proposed method. Therefore, it concludes that a dedicated layer-aware 3D partitioning algorithm, like our method and *iLap*, should be regarded as one of the essential components in the sophisticated 3D IC design flow. As compared with *iLap*, our method can further improve TSV count by 3% with about 1.5 times faster execution time.

The results with k=8 reported in Table 2 show that our method outperforms *hMetis* and *iLap* by about 39% and 6% on average in TSV count, respectively. The results also imply that the improvement can slightly increase as the design complexity grows to higher levels (i.e., *k* increases). The experimental results suggest that our method tends to generate a smaller footprint of a 3D chip implementation. Meanwhile, a lower TSV count is very helpful in designing 3D regular logic structures such as 3D FPGAs since the number of prefabricated inter-layer TSVs is fixed.

C. Analyses of Runtime Efficiency

Regarding the runtime efficiency issue, Table 3 and Table 4 give the runtime of 14 test cases in second when the number of layers is set to 4 and 8, respectively. We focus only on the deterministic phase here since only a few TSVs have been reduced in the stochastic phase. It is evident that *hMetis* is very time-efficient. The runtime required by our method grows logistically as the number of layers increases. It is mainly because the number of stages inside the deterministic phase of proposed algorithm also grows

Table 3. Runtime results with *k*=4.

4 layers	Execution time(s)				Normalized to hMetis		
Design	Our	iLap	hMetis	EX-hMetis	Our	iLap	EX-hMetis
Tseng	0.24	0.41	0.20	0.20	1.23	2.11	1.01
Diffeq	0.34	0.53	0.23	0.23	1.44	2.26	1.00
Des	0.41	0.70	0.39	0.40	1.04	1.78	1.02
Bigkey	0.17	0.33	0.15	0.15	1.10	2.17	1.02
Frisc	0.72	0.94	0.46	0.47	1.57	2.05	1.03
elliptic	0.56	0.79	0.35	0.36	1.63	2.30	1.05
pdc	1.10	1.50	0.68	0.70	1.61	2.19	1.02
fft128	0.54	0.78	0.43	0.45	1.23	1.79	1.04
s38417	1.03	1.55	0.81	0.83	1.28	1.92	1.02
s38584.1	1.21	1.90	0.89	0.91	1.36	2.13	1.02
clma	1.69	2.45	1.26	1.28	1.34	1.94	1.02
cfft	1.53	2.17	1.15	1.18	1.34	1.89	1.03
aqua	5.23	6.69	2.76	2.87	1.90	2.43	1.04
video	8.15	10.72	4.34	4.59	1.88	2.47	1.06
Average					1.42	2.10	1.03

Table 4. Runtime results with *k*=8.

8 layers	Execution time(s)				Normalized to hMetis		
Design	Our	iLap	hMetis	EX-hMetis	Our	iLap	EX-hMetis
Tseng	0.41	1.44	0.30	4.15	1.37	4.79	13.82
Diffeq	0.53	1.64	0.38	5.92	1.38	4.27	15.41
Des	0.64	2.06	0.60	7.47	1.08	3.47	12.56
Bigkey	0.32	1.22	0.26	7.07	1.24	4.67	27.09
Frisc	0.94	2.53	0.66	14.40	1.42	3.83	21.85
elliptic	0.74	3.81	0.53	14.42	1.38	7.13	27.01
pdc	1.50	4.04	0.93	21.36	1.61	4.33	22.92
fft128	0.73	2.35	0.66	15.46	1.10	3.53	23.28
s38417	1.38	4.55	1.25	25.98	1.10	3.63	20.75
s38584.1	1.59	5.63	1.37	28.13	1.15	4.10	20.47
clma	2.32	7.83	1.97	37.27	1.18	3.97	18.91
cfft	2.00	6.15	1.67	54.04	1.20	3.68	32.38
aqua	6.40	16.86	3.91	204.54	1.64	4.31	52.32
video	10.70	27.34	6.18	441.15	1.73	4.43	71.42
Average					1.33	4.30	27.16

logistically as the number of layers increases. Since *EX-hMetis* has to check all possible layer permutations to find out the best one, the required runtime is thus exponential to the number of layers. The runtime required by *iLap* grows linearly because the number of invocations for multi-way partitioning inside *iLap* also grows linearly as the number of layers increases. Hence, we can conclude that our proposed method can take advantages of modern parallel computing systems and maintains the runtime scalability in the future large-scaled designs.

IV. Conclusion

In this paper, we present a two-phased parallel layer-aware partitioning algorithm for TSV minimization in 3D structures. The first phase utilizes a two-way min-cut partitioning engine with OpenMP multi-core framework to get the initial solution. Then the second phase further reduces the number of TSV by a parallel simulated annealing approach on the many-core GPGPU computing system. The experimental results clearly demonstrate that our proposed method is capable of reducing total TSV count by about 39% compared to layer-unaware *hMetis*. It also achieves a 6% improvement over the iterative layer-aware partitioning algorithm, *iLap*, with 3.2X runtime enhancement. Consequently, since the parallel nature of proposed method, we are confident that it can generate better TSV-minimized solutions for reasonably large designs in an acceptable runtime.

References

[1] International Technology Roadmap for Semiconductor. Semiconductor Industry Association, 2009.

[2] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: a novel chip design for improving deep submicron interconnect performance and systems-on-chip integration," *Proc. IEEE*, vol. 89, no. 5, pp. 602–633, 2001.

[3] A. W. Topol, D. C. La Tulipe, L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini and M. Ieong, "Three-dimensional integrated circuits,"

[4] *IBM J. of Research and Development*, vol. 50, no. 4/5, pp. 491–506, Jul./Sep. 2006.

[5] G. Philip, B. Christopher, and P. Ramm, Handbook of 3D integration. Wiley-VCH, 2008.

[6] C. Ferri, S. Reda, and R. I. Bahar, "Parametric yield management for 3D ICs: models and strategies for improvement," *J. Emerging Technologies in Computing Systems*, vol. 4, no. 4, Article ID 19, Oct. 2008.

[7] M. Pathak, Y.-J. Lee, T. Moon, and S. K. Lim, "Through-silicon via management during 3D physical design: when to add and how many?" *Proc. Int'l Conf. on Computed-Aided Design*, pp. 387–394, 2010.

[8] I. H.-R. Jiang, "Generic integer linear programming formulation for 3D IC partitioning," *22nd IEEE Int'l SOC Conf.*, pp. 321–324, 2009.

[9] D. H. Kim, K. Athikulwongse, and S. K. Lim, "A study of throughsilicon-via impact on the 3D stacked IC layout," *Proc. Int'l Conf. on Computer-Aided Design*, pp. 674–680, 2009.

[10] Y. C. Hu, Y. L. Chung, and M. C. Chi, "A multilevel multilayer partitioning algorithm for three dimensional integrated circuits," *Proc. Int'l Symp. on Quality Electronic Design*, pp. 483–487, 2010.

[11] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," *Proc. Design Automation Conf.*, pp.175–181, 1982.

[12] C. Ababei, H. Mogal, and K. Bazargan, "Three-dimensional place and route for FPGAs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1132–1140, Jun. 2006.

[13] K. Siozios, A. Bartzas, and D. Soudirs, "Architecture-level exploration of alternative interconnection schemes targeting to 3D FPGAs: a software-supported methodology," *Int'l J. of Reconfigurable Computing*, vol. 2008, Article ID 764942, 2008.

[14] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Trans. on VLSI Systems*, vol. 7, no. 1, pp. 69–79, Mar. 1999.

[15] Y. S. Huang, Y. H. Liu and J. D. Huang, "Layer-Aware Design Partitioning for Vertical Interconnect Minimization," *Proc. of IEEE Computer Society Annual Symposium on VLSI*, pp. 144–149, Jul. 2011.

[16] W. J. Chen, H. K. Kuo, T. H. Chiu and B. C. Lai, "FDPrior: A Force-Directed Based Parallel Partitioning Algorithm for Three Dimensional Integrated Circuits on GPGPU," *Proc. of IEEE International Symposium on VLSI Design, Automation, and Test*, pp. 1–4, Apr. 2011.

[17] S. Yang, "Logic synthesis and optimization benchmarks user guide," Technical Report 1991-IWLS-UG-Saeyang, Microelectronics Center of North Carolina, 1991.

[18] http://www.eecs.berkeley.edu/~alanmi/benchmarks/altera/old/altera12_blif_baf.zip.