Finding System-Level Information and Analyzing its Correlation to FPGA Placement

Farnaz Gharibian and Lesley Shannon
School of Engineering Science
Simon Fraser University
Email: fga7,lshannon@sfu.ca

Peter Jamieson
Department of Electrical and Computer Engineering
Miami University
Email: jamiespa@muohio.edu

***Abstract*—One of the more popular placement algorithms for Field Programmable Gate Arrays (FPGAs) is called Simulated Annealing (SA). This algorithm tries to create a good quality placement from a flattened design that no longer contains any high-level information related to the original design hierarchy. Unfortunately, placement is an NP-hard problem and as the size and complexity of designs implemented on FPGAs increases, SA does not scale well to find good solutions in a timely fashion. As modern FPGAs can be used to implement Systems- and Networks-on-Chip, designers are required to spend an increasing amount of time waiting for place and route tools to complete that is not being matched by an increase in the power of computing work stations.**

In this paper, we investigate if system-level information can be reconstructed from a flattened netlist and evaluate how that information is realized in terms of its locality in the final placement. If there is a strong relationship between good quality placements and system-level information, then it may be possible to divide a large design into smaller components and improve the time needed to create a good quality placement. Our preliminary results suggest that the locality property of the information embedded in the system-level HDL structure (i.e. "module", "always", and "if" statements) is greatly affected by both the designer and the design itself. A reconstructive algorithm, called affinity propagation, is also considered as a possible method of generating a meaningful coarse grain picture of the design.

I. Introduction

Field Programmable Gate Arrays (FPGAs) have gained popularity as an implementation platform as their logic density has grown, allowing them to implement complete Systems-on-Chip (SoCs). During the development of an automated Computer Aided Design (CAD) flow for FPGAs, it was recognized that the placement problem could only be solved in Non-Polynomial (NP) time [1]. To this end, CAD designers leveraged heuristics to obtain good quality placement solutions in a timely fashion.

One of the more popular heuristic placement algorithm implementations for FPGAs is called Simulated Annealing (SA). It tries to create a good quality placement from a flattened design that no longer contains any system-level information from the original design hierarchy. As the size and complexity of FPGA designs increases, SA does not scale well; in fact, the time designers are required to wait for the successful place and route of a design using commercial CAD tools is reportedly not being alleviated by the gained computing power in new work stations [2].

The poor scaling of the placement algorithm's run time is in part due to the fact that the clock speed of modern day processors does not increase significantly with each new processing technology node due to power consumption. Instead, modern day processors rely on increasing the number of processing cores to improve task execution through parallelism. This need to leverage parallelism to reduce run time creates a second problem as placers must generate deterministic solutions, independent of the number of processing cores. Based on this requirement and the typically global approach taken by placement algorithms (including SA), multi-threaded implementations of placers typically have problems with the lack of spatial locality for their data; therefore, these parallelization techniques do not dramatically improve performance [2] [3].

In this paper, we present ***a study to determine the quality and types of system-level information that may be reconstructed by SA in the final placement***; to quantify the quality of this reconstruction, we use Manhattan Distance (MD) as a measure of locality. The specific contributions are:

- An analysis that suggests that the quality of the information embedded in the high-level circuit structure (i.e. *module*, *always*, and *if* statements) is greatly affected by both the designer and the design itself.
- How clustering algorithms, such as affinity propagation [4], can reconstruct system-level information that may be used to create coarse grain components to provide the flexibility needed for multi-phased placement.
- A method of evaluating the quality of system-level information in relation to placement using MD and the distribution of system-level granular blocks. This methodology allows new reconstructive algorithms to be easily analyzed and compared to one another.

If some general relationship between good quality placements and system-level information exists, then it is possible that dividing a large design into smaller components may improve the speed with which a good quality placement is obtained as we could better leverage new multicore processing platforms. Ideally, the general placement of a design using SA can be somewhat intuited from the cost function it uses to evaluate the quality of a placement: minimize the wire lengths of connected blocks with the objective of minimizing the critical path. However, as SA relies on randomly swapping clusters, there is no direct link between the original design structure

and the structure of the final placement.

The remainder of this paper is organized as follows. Section II discusses the related work including placement algorithms for FPGAs and using some form of design partitioning/clustering in combination with placement. Section III describes what we mean by "system-level information" and why partitioning designs into coarse grain components guided by system-level information may be valuable for reducing run time and even improving placement quality. Section IV summarizes the CAD flow used to obtain our data. Section V presents our experimental results and Section VI concludes the paper, with thoughts as to possible directions for future work.

II. Background

The goal of placement in the FPGA CAD flow is to find a valid placement for each logic block while trying to minimize the critical path; this objective is typically represented as a cost function used to minimize the total length of interconnect required. There are two main approaches to placement: analytical and heuristic algorithms. Analytic placers, such as quadratic placement [5] [6], are commonly used in commercial FPGA CAD flows to provide an initial high-level placement. They generally complete in less time than other heuristics but do not guarantee a legal placement as the final placement might have overlapping logic blocks [5]. Instead, they are generally used to provide an initial placement for larger designs that is fine tuned using heuristic algorithms (e.g. SA) to remove the overlap, resulting in a more scalable run time for placers as design complexity increases.

Simulated Annealing [7] (SA) is oftenused for at least a portion of an FPGA's placement. Unfortunately, SA requires significant computation time to produce good placements. SA swaps random clusters to move through the design space, in combination with a "cooling table" that allows the algorithm to select the occasional "poor" placement (in terms of its cost function) to perform some hill climbing. Different techniques have been used to improve SA's run time, some of which have been included in the version of SA incorporated into the academic CAD flow VPR [8]. Other techniques for reducing the run time of SA rely on:

- some form of partitioning/clustering the design into groups of clusters (which we call ***super-clusters***) before running the placer [9],
- quickly creating a *good* initial placement to reduce the iterations required to find the final placement [10], and
- using multi-threading techniques that allow the placer to leverage the increased processing power available on multicore processors [11], [2].

Although our long term research objectives include creating a placement algorithm that scales better in performance for multicore processors, the current study focuses on a *good* placement's final structure in relation to the actual design. As such, the remainder of this discussion on SA focuses on previous work reducing the actual number of iterations of SA using partitioning/clustering or a good initial placement. Work on multi-threading techniques is complementary to this work as it instead focuses on completing SA iterations more quickly.

In a Two Stage Simulated Annealing (TSSA) placer, the early randomizing actions occurring at the highest temperatures of the cooling schedule are replaced by a faster heuristic with a traditional, monotonically-decreasing temperature regime to create an initial placement [12]. The SA phase leverages this "good" initial placement solution to start at a lower temperature than is normally required to achieve the desired solution quality for the problem being considered. For this reason, the SA phase of the TSSA system is often referred to as the low temperature annealing phase [12].

Recently, techniques to reduce ASIC placer run times have also been studied for FPGA placers due to the growth in their design capacity. Bian et al. [13] investigated three ASIC placement algorithms that include: partition-based [14], a multilevel approach [15] and a quadratic analytical [16] placer in conjunction with SA. Their results show that as the number of clusters increases, VPR's SA takes more time than these other placement methods. The authors also introduce a placement technique based on maximum-bipartite matching [13]. They report that their method can produce comparable results to VPR given generous white space.

Partitioning/Clustering methods to construct high-level information: The Ultra-Fast Placement algorithm aims to improve the runtime of VPR's SA placer by initially performing multi-level clustering [9]. To clarify our discussions, we define *clusters* as logic blocks on an FPGA, whereas we define *super-clusters* as groupings of one or more clusters in a design. In Ultra-Fast, the super-cluster sizes at each level are *fixed* to facilitate the exchange of clusters during the swapping moves in SA. The super-clusters are grown based on a connectivity-based scoring function determined by two components: the strength of connections between the blocks and the number of nets that are absorbed if a block is merged into the super-cluster. Each level is performed in two phases: a phase to build a good initial placement followed by a low temperature SA phase. Although we are interested in using system-level information for clustering algorithms and multi-phased placement, we do not assume a fixed number of clusters per super-cluster (like the Ultra-Fast Placement algorithm [9]) or a fixed number of super-clusters (like the K-clustering algorithms [17]

Design Level Information: FPGA placements based on SA use random initial placements and do not consider information embedded in the original design as circuits are typically flattened before SA is run. Some previous work in ASIC placers suggest that high-level information and design hierarchy should be considered during both clustering [18] and placement [19]. Previously, floorplanning (or hierarchical) approaches to placement, based on the design's hierarchy as specified in its RTL have been introduced [20], [21]. For example, Emmert et al. [20] start by decomposing the FPGA device into an array of placement bins with the same physical dimensions. Then a macro-based netlist of soft and hard macros targeted to the device is created. Finally, it groups

the macros into clusters that can be mapped into bins.

III. System-level Information

The idea of trying to replace SA's time consuming step of moving from a random placement to a reasonable one via random swaps with a good initial placement is discussed by Grover [12]. This TSSA work is based on the hypothesis that: SA wastes effort using random swaps to generate its first reasonable solution; instead, it would be better to generate a good initial placement utilizing more computationally efficient methods and apply SA at a low temperature. We propose using *System-level Information* to guide the generation of this initial placement, where we define *System-level Information* as: a granular grouping of the design based on connectivity and/or structure that may be obtained either from its graphical representation or its original HDL (design-level) description. We are particularly interested in determining if this information can be used to create coarse grain substructures to guide initial placements for heuristics, and potentially lead to a more scalable placement solution multicore architectures.

To evaluate how useful this approach might be, we first assume that as SA generates good quality placements, it uncovers substructures that have locality. We propose that these "uncovered" substructures fall into two categories: 1) those that are tied to the original design's structure and can be found using system-level information, and 2) those that are not apparent from the original design but truly "uncovered" by SA's random swaps. The remainder of this paper focuses on the first category of substructures, investigating what types of system-level information SA "reconstructs" in its final placement and the quality of reconstruction. To quantify this metric, we utilize the Manhattan Distance (MD) of the set of clusters that implement the substructure, where a smaller MD reflects greater locality for the substructure and thus a better reconstruction. If certain types of information have poor locality in an SA placement, it may not be as useful in creating coarse grain substructures for an initial placement.

In this study, we investigate the locality of system-level information within a final SA placement using two approaches. The first is based on the structure of the source HDL, where we try to determine how hierarchical HDL structures (specifically, *module*, *always*, and *if* blocks) are reflected in the final placement. Our method of detecting HDL components and generating results is reported in Section IV. The second approach uses a reconstructive algorithm, called affinity propagation [4], to analyze the connectivity graph of the netlist after logic has been packed into clusters. It detects relationships that determine what clusters should be considered part of the same coarse grain substructure.

Affinity propagation is a clustering algorithm that is applied to an input graph to group clusters based on a defined similarity factor and generate a set of super-clusters. These super-clusters are created via a series of message passing steps, where nodes in the graph communicate their attraction to other nodes. The number and size of the super-clusters created by the algorithm depends on the structure of the graph and a similarity factor that describes how every two nodes in the graph are initially related to one another. We have implemented two versions of affinity propagation that use slightly different metrics to create the initial similarity factors. The first (AFFIN1) assumes that the similarity of two nodes in the graph (representing circuit components) is higher for shorter graph distances. For example, if two nodes are connected directly to each other, the assumption is that they should be placed closer together on the FPGA. However, for longer path lengths, the similarity between nodes is lower as they do not need to be placed close together. Our second version of affinity (AFFIN2) creates similarities based on three properties: common connections, proximity, and fan out. Specifically, a node is considered more similar to another node by this version of the algorithm if: they have more than one connection, are close in proximity, and their respective fanout is high. For example, unlike AFFIN1, AFFIN2 is more likely to closely place two nodes with many connections than two nodes with closer proximity yet having only one connection between them.

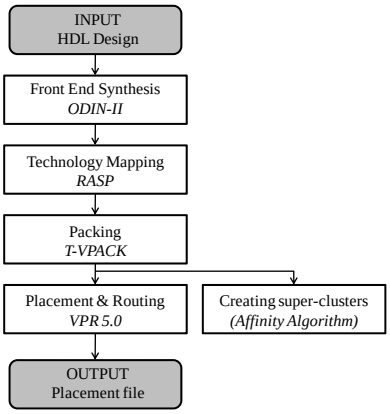

Fig. 1. FPGA CAD flow used in this research

IV. Experimental Methodology

Figure 1 shows the FPGA CAD flow used in this work to determine how structural information from both the HDL design and the circuit netlist are related to the SA's final placement. A Verilog HDL design is inputted to Odin II [22], which creates a flattened netlist consisting of system-level I/Os, logic gates, flip-flops, and hard circuits (e.g. multipliers/memories). This netlist is passed into RASP [23] for logic optimization and mapping to Look-Up Tables (LUTs). RASP was chosen specifically for this project as it maintains the majority of design-level naming in the netlist, allowing us to measure the MD of various structures (i.e. *module*, *always*, and *if*) in the final placement. Next, T-VPack [24] takes the output from RASP and packs the LUTs and registers into clusters. The output of T-VPack is then placed and routed onto an FPGA architecture described by the parameters summarized in Table I using VPR 5.0 [25]. Our two versions of affinity propagation (AFFIN1 and AFFIN2) also use T-VPACK's output to generate super-clusters based on their respective similarity factors outlined in

TABLE I
THE FPGA ARCHITECTURAL PARAMETERS

Parameter	W	N	K	F_{cin}	F_{cout}	F_s	routing
Value	20% larger than minimum	10	4	0.18	0.1	3	uni-directional

TABLE II
RESULTS AFTER VPR COMPLETES PLACE & ROUTE

Benchmark	Number of IO	Number of Clusters	Grid Size	Std Dev	GMEAN
cfc18	111	627	26*26	5.08	95.66
cft8	69	1031	33*33	5.59	81.58
desa	190	265	17*17	4.39	111.59
iir1	59	142	12*12	1.02	54.75
oc54	140	530	24*24	0.64	24.60
pajf	103	166	13*13	14.44	233.50
rsd2	32	496	23*23	0.88	57.61
dconvert	258	445	22*22	0.74	20.28
dsystemC	162	409	21*21	0.39	17.98
glue2	40	43	7*7	3.13	91.75
rsd1	20	192	14*14	1.23	131.14

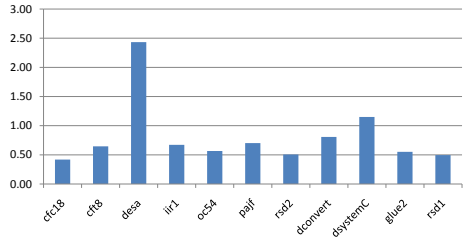

Fig. 2. Ratio of the Global Manhattan Distance: AFFIN2/AFFIN1

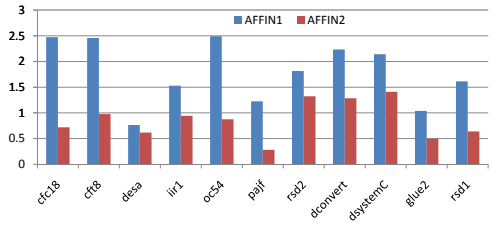

Fig. 3. Average local MD per cluster for AFFIN1 and AFFIN2

Section III.

We use the Manhattan Distance (MD) of a super-cluster's bounding box in the final SA placement to measure the locality of a super-cluster. Specifically, a super-cluster's MD is equal to half the perimeter of the bounding box of all the clusters in that super-cluster. The clusters in the super-clusters (generated using affinity propagation or from a benchmark's *module*, *always*, and *if* blocks) are located basically the same way - by searching the final placement for the locations of all clusters identified in a super-cluster. However, whereas the affinity propagation algorithm outputs a list of super-clusters and their component clusters, we use the design-level naming scheme generated by Odin II and passed through the CAD flow to locate individual *module/always/if* instantiations.

V. RESULTS

In this section, we summarize the outputs of the VPR place and route and discuss how certain types of system-level information correlate to the final placement.

A. VPR results

Table II shows the VPR statistics for 11 of the larger open source benchmarks available in Verilog HDL. Column 1 contains the names of the benchmarks, and Columns 2 through 4 list the number of I/Os, the number of clusters, and the array size for each benchmark respectively. Columns 5 and 6 show the standard deviation and geometric mean of the operating frequency of the benchmarks in MHz over ten random seeds.

B. Affinity and HDL Placement Correlation Results

Table III summarizes the results of our first experiment using AFFIN1 (top) and AFFIN2 (bottom) to reconstruct high-level information into super-clusters. First, we run both AFFIN1 and AFFIN2 to create the super-clusters and then find the MD of these super-clusters in the ten placements generated for each benchmark to evaluate our affinity propagation clustering algorithms. Columns 2 shows the number of super-clusters constructed by each version of the algorithm. Columns 3 through 6 gives the distribution of clusters in the super-clusters where the 1st two columns list the minimum and maximum number of clusters in a super-cluster and Columns 5 and 6 show the standard deviation and the geometric mean. Columns 7 through 10 list the distribution of super-cluster MDs (Local MDs) in a similar format. Columns 11 and 12 give the standard deviation and geometric mean of the Global MD generated by totalling the MDs of each super-cluster. Column 13 contains the ratio of the average Local MDs for the super-clusters (Column 10) divided by the average number of clusters per super-cluster (Column 6).

Table IV shows the results of our second experiment, measuring how the *module*, *always*, and *if* HDL constructs relate to the final placements generated by the SA placer (for ten random seeds). The results for all three constructs are reported using the same format. Subcolumn 1 states the total number of each construct located in the source HDL versus the total number that we could successfully map to clusters on the final design. The number of constructs that could be successfully mapped to clusters is less than the total as certain (unknown) logic optimizations in RASP remove high-level naming information from the output netlist, meaning that these high-level constructs are "lost" in the final placement. Therefore, the number and percentage of clusters successfully mapped to super-clusters are listed in Subcolumns 2 and 3 respectively. Finally, Subcolumns 4 and 5 list the standard deviation and geometric mean of the number clusters per HDL construct-based super-cluster.

C. Data Analysis

An examination of the results shown in Table III indicates that AFFIN2 generally creates fewer super-clusters (excluding *pajf*) with a more even distribution of clusters and a larger number of clusters per super-cluster on average. In contrast, the Local MD of super-clusters in AFFIN2 is also generally

TABLE III
RESULTING SUPER-CLUSTERS USING TWO DIFFERENT AFFINITY PROPAGATION METRICS.

AFFIN1												
Benchmark	**Super-clusters**	**Clusters**				**Local MD**				**Global MD**		**Local MD/Clusters**
		min	max	STD	Gmean	min	max	STD	Gmean	STD	Mean	
cfc18	65	1	179	21.52	6.71	0	49	8.6	16.6	23.44	1041.8	2.47
cft8	91	3	516	53.51	5.84	2	63	12.07	14.35	25.34	1332.6	2.46
desa	6	6	231	91.53	12.22	5	31	10.61	9.33	1.78	57.5	0.76
iir1	15	5	51	11.57	7.32	6	22	4.99	11.2	6.28	169.1	1.53
oc54	43	3	247	36.72	6.88	2	45	7.88	17.14	12.15	737.4	2.49
pajf	5	3	148	64.18	8.82	3	24	7.92	10.8	2.66	54.2	1.22
rsd2	50	3	209	28.78	6.03	3	43	7.51	10.94	14.33	563.3	1.81
dconvert	38	3	228	36.08	6.16	3	41	8.35	13.76	12.64	520.5	2.23
dsystemC	37	3	222	35.68	5.48	2	39	8.16	11.73	14.61	414.7	2.14
glue2	6	3	21	6.94	5.46	3	11	3.14	5.67	3.38	34.1	1.04
rsd1	19	3	88	19.01	6.2	2	26	6.18	10	9.69	190.8	1.61

AFFIN2												
Benchmark	**Super-clusters**	**Clusters**				**Local MD**				**Global MD**		**Local MD/Clusters**
		min	max	STD	Gmean	min	max	STD	Gmean	STD	Mean	
cfc18	13	18	75	17.67	44.8	13	46	8.35	32.23	9.52	435.8	0.72
cft8	35	12	131	21.78	25.06	9	62	15.22	24.51	21.26	860.9	0.98
desa	8	17	64	19.45	29	11	22	3.72	17.88	6.66	139.9	0.62
iir1	8	7	31	7.7	16.17	5	21	5.28	15.25	5.82	113.5	0.94
oc54	18	9	52	12.89	26.46	13	31	5.04	23.17	6.11	417.4	0.88
pajf	118	1	40	3.63	1.07	0	15	1.77	0.3	2.67	38	0.28
rsd2	16	5	204	52.1	13.99	4	36	10.33	18.5	15.42	284.8	1.32
dconvert	25	6	47	12.47	14.16	4	39	9.83	18.2	13.94	420	1.29
dsystemC	27	4	32	8.36	12.84	3	37	9.83	18.11	12.94	476.9	1.41
glue2	2	14	29	10.61	20.15	9	11	1.41	10	0.92	18.8	0.50
rsd1	6	11	75	24.67	25.57	10	23	4.8	16.33	9.81	94.6	0.64

TABLE IV
RESULTING SUPER-CLUSTERS BASED ON *module*, *always*, AND *if* STATEMENT DECLARATIONS.

Bench-mark	**ALWAYS**					**IF**					**MODULE**				
	blocks	**clusters**	**%Map**	**STD**	**Geo**	**blocks**	**clusters**	**%Map**	**STD**	**Geo**	**blocks**	**clusters**	**%Map**	**STD**	**Geo**
cfc18	114:74	166	0.26	18.57	458	114:0	0	0.00	0.00	0	1:1	627	1.00	0.00	49
cft8	387:91	285	0.28	56.27	763	387:2	10	0.01	1.17	7	1:1	911	0.88	0.32	63
desa	11:1	32	0.12	2.87	25	0:0	0	0.00	0.00	0	11:3	202	0.76	4.12	71
iir1	3:2	21	0.15	2.12	23	3:2	21	0.15	2.12	23	5:5	142	1.00	2.62	74
oc54	22:18	399	0.75	9.34	364	26:15	171	0.32	11.80	302	8:8	464	0.88	3.80	184
pajf	7:5	166	1.00	1.34	65	13:3	85	0.51	1.34	41	4:3	166	1.00	0.52	34
rsd2	42:22	264	0.53	5.19	237	17:17	143	0.29	6.74	173	23:7	267	0.54	4.68	175
dconvert	1:1	300	0.67	0.32	41	1:1	300	0.67	0.32	41	1:1	445	1.00	0.00	41
dsystemC	1:1	289	0.71	0.00	39	1:1	289	0.71	0.00	39	1:1	409	1.00	0.00	39
glue2	4:3	41	0.95	1.05	16	13:6	40	0.93	0.48	12	1:1	41	0.95	0.48	12
rsd1	54:25	174	0.91	8.44	155	21:20	121	0.63	6.43	102	31:7	180	0.94	3.84	110

larger than that of AFFIN1, which may suggest poorer locality of AFFIN2 super-clusters in the SA placement. However, Figure 2 highlights how the Global MD using AFFIN 2 is generally less than AFFIN1 (excluding *desa* and *dsystemC*) using the relation AFFIN2/AFFIN1, suggesting that the larger Local MDs are mostly due to the larger number of clusters in AFFIN2's super-clusters. This argument is strengthened by looking at the average Local MD per cluster ratio shown in Figure 3. As the MD of a single cluster is zero, this metric decreases the more tightly packed clusters are within a super-cluster mapping. For all our benchmarks, AFFIN2 has the lower ratio.

The correlation between HDL constructs and final placements is harder to interpret as shown in Table IV. First of all, the loss of clusters due to loss of naming information during logic optimization (as discussed in Section V-B) results in only "samples" of these HDL-construct-based super-clusters being visible for analysis. Furthermore, in the cases of the *always* and *if* blocks, these are generally small percentages of the design; in fact, only the bottom four entries of Table IV can provide super-cluster mappings that represent at least 60% of

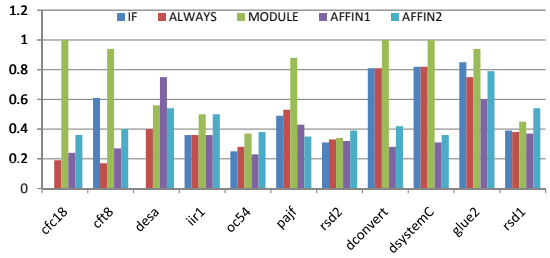

Fig. 4. Minimum MD over actual MDs in super-clusters

the design. Of these four benchmarks, *dconvert* and *dsystemC* highlight an important consideration for using HDL-constructs to generate super-clusters: they only have one *module*, *always*, and *if* block that encapsulates most of the clusters in the design. Therefore, using HDL-constructs to generate a coarse grain mapping of the design requires that the design be large enough and that the designer has used these constructs to encapsulate sufficient design hierarchy (i.e. *modules*) and/or control structures (i.e. *always*, *if*) that multiple meaningful super-clusters can be generated.

Since the HDL-construct-based super-clusters considered

here do not generally represent a significant portion of their overall designs, Global MD comparisons with the results from affinity propagation are not very meaningful. Instead, we need a metric that considers the average Local MD of the super-clusters found in all cases, normalized to reflect the value of denser cluster packing. Specifically, for each super-cluster in a benchmark, we determine its minimum theoretical MD (based on the number of clusters) and divide this by the actual MDs that resulted in the SA placement. We then found the geometric mean of this measure over all super-clusters in a benchmark and plotted this data for all five super-cluster generation options in Figure 4. The resulting data can be interpreted as follows: this ratio ranges from zero to one and approaches one as the super-clusters are placed on the FPGA with close to the theoretical minimum MD.

At first glance, these results suggest that using *module* instantiations provide a good basis for super-cluster generation. However, excluding the *paij* benchmark, all values over 0.6 occur when there is only a single module in the design. Similar results can be seen for the *always* and *if*, (excluding the *glue* benchmark). Nevertheless, while these results suggest that HDL constructs cannot be used to generate super-clusters for these benchmarks, the outliers suggest that if our designs were much larger, these constructs might be useful. Still, this would also be dependent on the nature and the structure of the design. Although both versions of affinity propagation have lower ratio values using this metric, we again see that generally AFFIN2 does better than AFFIN1 in producing super-clusters that reflect better spatial locality within SA's placement. We expect that using the metrics for comparing the quality of super-cluster locality within a placement illustrated in Figures 3 and 4, we can evaluate and better tune the similarity factors to construct better super-clusters for an initial coarse grain placement. They can also be used to assess other constructive algorithms.

VI. Conclusion and Future Work

In this paper, we have summarized a study of how system-level information may be reconstructed by SA in its final placement. We presented a method of evaluating the locality of coarse grain substructures within a placement along with an analysis of how specific types of system-level information correlate to an SA's final placement. Our results suggest that system-level information can be used to find coarse grain mappings that could improve placement algorithms. Finally, we described and implemented two different versions of the affinity propagation algorithm and demonstrated how the similarity factors (AFFIN2) can be tuned to obtain better system-level information with a better relation to their final placement on an FPGA.

Currently, we are looking into methods of ensuring that *all* design-level information is maintained throughout the entire CAD flow. We are also examining other possible structures (i.e. Buses, Datapath) for possible correlations in the final placement to determine which structure(s) has the best correlation in the final placement. Future work includes investigating new possible similarity factors that may result in coarse grain structures with better locality on a SA placement and trying to use system-level information to generate an initial coarse grain placement.

Acknowledgment

The authors would like to acknowledge NSERC's funding of this research and thank Mr. Kevin Chung for his help and advice.

References

[1] W. E.Donath, "Complexity theory and design automation," in *In Proc. of the 17th Design Automation Conference*, 1980, pp. 412–419.

[2] A.Ludwin *et al.*, "High-quality, deterministic parallel placement for fpgas on commodity hardware," in *Proc. of the 16th Int'l ACM/SIGDA Symp. on FPGAs*, 2008, pp. 14–23.

[3] J.Chandy *et al.*, "Parallel simulated annealing strategies for vlsi cell placement," in *9th Int'l Conference on VLSI Design*, 1996, pp. 37–42.

[4] B. J.Frey *et al.*, "Clustering by passing messages between data points,," in *Science*, vol. 315, Feb. 2007, pp. 972–976.

[5] B. M.Riess *et al.*, "Speed: Fast and Efficient Timing Driven Placement," in *IEEE Int'l Symp. on Circuits*, 1995, pp. 377–380.

[6] K.Vorwerk *et al.*, "Engineering details of a stable force-directed placer," in *Proc. of the 2004 IEEE/ACM Int'l Conf on CAD*, 2004, pp. 573–580.

[7] S.Kirkpatrick *et al.*, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[8] V.Betz *et al.*, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[9] Y.Sankar *et al.*, "Trading quality for compile time: ultra-fast placement for fpgas," in *ACM/SIGDA Int'l Symp. on FPGAs*, 1999, pp. 157–166.

[10] J. M.Varanelli *et al.*, "A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems," *Computers and Operations Research*, vol. 26, no. 5, pp. 481–503, 1999.

[11] M.Haldar *et al.*, "Parallel algorithms for fpga placement," in *10th Great Lakes Symp. on VLSI*, 2000, pp. 86–94.

[12] L. K.Grover, "Standard cell placement using simulated sintering," in *Proc. of the 24th ACM/IEEE DAC*, 1987, pp. 56–59.

[13] H.Bian *et al.*, "Towards scalable placement for fpgas," in *Proc. of the 18th annual ACM/SIGDA Int'l symp. on FPGAs*, 2010, pp. 147–156.

[14] A. E.Caldwell *et al.*, "Optimal partitioners and end-case placers for standard-cell layout," *Int'l Symp on Physical Design*, pp. 90–96, 1999.

[15] T. F.Chan *et al.*, "Multilevel optimization for large-scale circuit placement," in *IEEE/ACM Int'l Conf on CAD*, 2000, pp. 171–176.

[16] N.Viswanathan *et al.*, "Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," *Proc. of the 1997 Int'l Symp. on Physical design*, 2004.

[17] J. B.MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symp. on Mathematical Statistics and Probability*, vol. 1. University of California Press, 1967, pp. 281–297.

[18] D.Behrens *et al.*, "Circuit partitioning using high-level design information," in *Conference on Integrated Design - process Technology*, 1996, pp. 259–266.

[19] Y. W.Tsay *et al.*, "Preserving hdl synthesis hierarchy for cell placement," *Proc. of the 1997 Int'l symp. on Physical design*, pp. 169–174, 1997.

[20] J. M.Emmert *et al.*, "A methodology for fast fpga floorplanning," in *Proc. of the 7th Int'l ACM/SIGDA Symp. on FPGAs*, February 21-23, 1999, pp. 47–56.

[21] R.Tessier, "Fast placement approaches for fpgas," *ACM Trans. on Design Automation of Electronic Systems*, vol. 7, no. 2, pp. 284–305, April 2002.

[22] P. A.Jamieson *et al.*, "Odin II - An Open-source Verilog HDL Synthesis Tool for Academic CAD Flows," in *Field-Programmable Custom Computing Machines, Annual IEEE Symp. on*, 2010.

[23] J.Cong *et al.*, "RASP: A General Logic Synthesis System for SRAM-Based FPGAs," in *ACM/SIGDA Int'l Symp. on FPGAs*, 1996, pp. 137–143.

[24] A.Marquardt *et al.*, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," in *ACM/SIGDA Int'l Symp. on FPGAs*, Monterey, CA, 1999, pp. 37–46.

[25] J.Luu *et al.*, "VPR 5.0: FPGA CAD and Architecture xploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling," in *ACM/SIGDA Int'l Symp. on FPGAs*, Feb 2009.