Maximizing Speed and Density of Tiled FPGA Overlays via Partitioning

Charles Eric LaForest
Department of Electrical and Computer Engineering
University of Toronto, Canada
laforest@eecg.toronto.edu

J. Gregory Steffan
Department of Electrical and Computer Engineering
University of Toronto, Canada
steffan@eecg.toronto.edu

***Abstract*—Common practice for large FPGA design projects is to divide sub-projects into separate synthesis partitions to allow incremental recompilation as each sub-project evolves. In contrast, smaller design projects avoid partitioning to give the CAD tool the freedom to perform as many global optimizations as possible, knowing that the optimizations normally improve performance and possibly area. In this paper, we show that for high-speed tiled designs composed of duplicated components and hence having *multi-localities* (multiple instances of equivalent logic), a designer can use partitioning to preserve multi-locality and improve performance. In particular, we focus on the lanes of SIMD soft processors and multicore meshes composed of them, as compiled by Quartus 12.1 targeting a Stratix IV EP4SE230F29C2 device. We demonstrate that, with negligible impact on compile time (less than $\pm$10%): (i) we can use partitioning to provide high-level information to the CAD tool about preserving multi-localities in a design, without low-level micro-managing of the design description or CAD tool settings; (ii) by preserving multi-localities within SIMD soft processors, we can increase both frequency (by up to 31%) and compute density (by up to 15%); (iii) partitioning improves the density and speed (by up to 51 and 54%) of a mesh of soft processors, across many building block configurations and mesh geometries; (iv) the improvements from partitioning increase as the number of tiled computing elements (SIMD lanes or mesh nodes) increases. As an example of the benefits of partitioning, a mesh of 102 scalar soft processors improves its operating frequency from 284 up to 437 MHz, its peak performance from 28,968 up to 44,574 MIPS, while increasing its logic area by only 0.85%.**

I. Introduction

Soft overlay architectures can ease design challenges on FPGAs by making them more software-programmable. Examples of such systems include VESPA [1], VEGAS [2], VENICE [3], iDEA [4], [5], Octavo [6], and others [7]–[11]. In general, overlays provide parallelism through "tiling" (duplicating in two dimensions) computing elements such as datapaths and soft processors. Our goals for this work are to discover the best practices for achieving (i) maximum operating frequency, and (ii) maximum compute density (the amount of possible work per unit area) for tiled soft overlays. Notably we explore tiled designs that operate at much higher than normal clock frequencies (400 to 500 MHz), exacerbating critical paths.

Tiled designs necessarily contain logically equivalent, duplicated circuitry across each tiled element. We refer to this circuitry as "multi-local" since it operates locally and identically in multiple instances across a design. Normally, to reduce logic usage, a CAD tool performs redundancy elimination to optimize multi-localities down to a single instance ("deduplication") and then fans-out its output to all the original locations. While generally beneficial on a small scale, deduplication may introduce new critical paths on a larger scale or in higher-speed circuits, for only a modest reduction in logic usage. Current methods to control deduplication involve manually micro-managing the design description or the CAD tool settings on a per-logic-node basis [12].

A. Partitioning

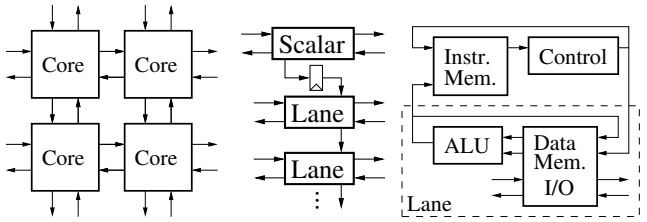
Partitioning refers to the *logical* division of a design into one or more sub-sections, usually at module boundaries, which then synthesize as separate netlists. The total design remains the same except for certain optimizations such as register retiming or (de)duplication, and boolean simplifications, which do not cross partition boundaries. In this study we find that partitioning provides a simple and effective method to provide high-level information to the CAD tool to preserve multi-localities during synthesis and thus avoid deleterious deduplication. *Merely partitioning the major datapaths of a tiled system suffices to prevent harmful global deduplication, while preserving beneficial local optimizations, for little cost in area.* Furthermore, this use of partitioning causes no significant changes in total CAD time, granting improved performance "for free" without increasing the design cycle time.

Rather than focus entirely on maximum operating frequency, we also consider the overall efficiency of the resulting designs in terms of computation per unit FPGA area per cycle—in other words, the *compute density*. A denser implementation is more desirable to replicate and tile into multicores. We demonstrate that partitioning improves density by increasing speed more so than area.

B. Floorplanning

Floorplanning describes the process of *spatially* preallocating areas on the FPGA device to sub-modules of a larger project, allowing them to evolve independently as sub-projects without encroaching on the placement (and thus, the timing) of other sub-modules. We find that floorplanning has no clear or predictable benefit to compute density, and always lowers compute density relative to the same design without a floorplan. *However, these results also show that a designer can tile an overlay without concern for the placement, proximity, or shape of each tile, letting the CAD tool find a good solution itself.*

Designers do currently use floorplanning and partitioning as project management techniques to enable incremental recompilation in larger, multi-part projects [13]–[15], but solely to preserve past CAD results, without any focus on improving performance directly. Similarly, designers leave smaller projects unpartitioned and free-form to give the CAD tool the freedom to perform as many global optimizations as possible, knowing that those optimizations normally improve performance and area.



(a) Mesh (b) Core (c) Scalar

Fig. 1: **The major modules used for experiments. Meshes (a) directly connect the I/O of Cores in a bidirectional North/South/East/West manner. Each Core (b) contains a Scalar processor with zero or more SIMD Lanes, all pipelined one instruction behind the Scalar processor for best performance. When Layered, each SIMD Lane may further pipeline the instructions to the next Lane. Each Scalar (c) processor contains a simple Control path and Data path with I/O memory-mapped into the Data Memory. Each SIMD Lane contains a single Data path.**

In this paper, we focus on tiled designs composed of SIMD lanes of soft processors, and multicore meshes built up from these. We construct these designs as extensions to the openly available Octavo[1] soft processor [6] whose operating frequency can reach up to the BRAM limit of 550MHz on Stratix IV FPGAs, and thus a good candidate for exploring the effects of tiled scaling and partitioning.

We explore the speed, area, and density of SIMD processors with 0 to 32 lanes, with and without partitioning, and find that partitioning preserves multi-locality, increasing speed and density significantly, and increasing area moderately. We try and extend these SIMD results by adding instruction pipeline stages *between* each SIMD lane to force a sequential dependency between each lane to prevent deduplication. Pipelining does provide some benefits, but can only crudely approximate the effects of partitioning and staggers the execution across SIMD lanes. Finally, we create rectangular meshes (with North, South, East, and West point-to-point bidirectional links) of soft processors, with and without SIMD lanes. We find that constructing high-speed meshes exposes different *unconnected* multi-localities, preserved with different partitioning, despite having the same underlying hardware as in the SIMD study. Partitioning meshes also increases their speed and density, with little area increase.

C. Contributions

Our main contribution is demonstrating that partitioning can preserve the multi-locality of tiled designs, without requiring detailed per-node micro-management, and without interfering with other beneficial optimizations or the place-and-route process. Partitioning also gives better results than having the CAD tool indiscriminately either globally remove duplicate logic or not remove any. In addition, we demonstrate that: floorplanning provides no predictable benefits to compute density, and generally lowers it; partitioning the major datapaths of SIMD and Mesh tiled soft processors improves speed and compute density, with the improvement increasing with the number of tiles and *without increasing the total CAD time*; and that we can force the CAD tool to preserve multi-locality by pipelining the paths between *connected* multiply-local logic instances, albeit also at the cost of staggered execution, and with no improvement to *unconnected* multi-localities.

[1]Available on GitHub at https://github.com/laforest/Octavo

II. Experimental Framework

Test Harness We place our designs inside a synthesis test harness designed to both: (i) register all inputs and outputs to ensure an accurate timing analysis, and (ii) to reduce the number of I/O pins to a minimum as larger designs will not otherwise fit on the FPGA. The test harness also avoids any loss of circuitry caused by I/O optimization. Shift registers expand single-pin inputs, while registered AND-reducers compact word-wide signals to a single output pin.

Synthesis We use Altera's Quartus 12.1 to target a Stratix IV `EP4SE230F29C2` FPGA device of the highest available speed grade. For maximum portability, we implement our designs in generic Verilog-2001, with some LPM[2] components. We configure the synthesis process to strongly favor speed over area and enable all relevant optimizations. To confirm the intrinsic performance of a design without interference from optimizations—such as register retiming, which can blur the distinction between the design under test and the test harness—we constrain the whole design under test to its own design partition, excluding the test harness. To explore the effects of duplicate register removal, we sometimes disable this specific optimization, denoted throughout as "nrdr" (No Removal of Duplicate Registers).

Place and Route We configure the place and route process to exert maximal effort at fitting and routing with only two constraints: (i) to avoid using I/O pin registers to prevent artificially long paths that would affect the clock frequency, and (ii) to set the target clock frequency to 550MHz, which is the maximum clock frequency specified for M9K BRAMs. Setting a target frequency higher than 550MHz does not improve results and could in fact degrade them: for example, a slower derived clock would aim towards an unnecessarily high target frequency, causing competition for fast routing paths.

Frequency We report the unrestricted maximum operating frequency (F_{max}) by averaging the results of ten place and route runs, each starting with a different random seed for initial placement. We construct the average from the slow-corner timing reports which assume a die temperature of 85° and a supply voltage of 900mV. Note that minimum clock pulse width limitations in the M9K BRAMs restrict the actual operating frequency to 550MHz, regardless of actual propagation delay. Reported F_{max} in excess of this limit indicates timing slack available to the designer.

Area and Density We also average the area over the ten random initial seeds. We measure area as the count of *equivalent Adaptive Logic Modules* (eALMs) in use, which include the equivalent silicon area of hard blocks such as M9K BRAMs (28.7 ALMs each) and DSP blocks (29.75 ALMs each) as reported by Wong *et al.* [16]. We calculate compute density as total Peak MIPS (over all datapaths) per 100 eALMs. By itself, a Stratix IV ALM roughly contains two 6-LUTs, two flip-flops, and two full-adders with carry-chain logic.

III. Floorplanning a Scalar Core

We initially explored floorplanning multiple tiles in separate and adjacent floorplans to preserve their multi-localities,

[2]*Library of Parametrized Modules* (LPM) describes hardware as modules instead of inferring it automatically from behavioral code.

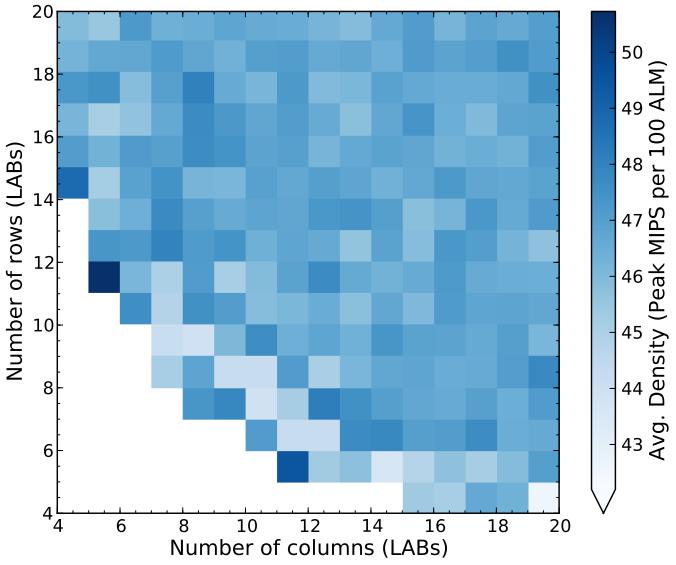


Fig. 2: **Heat map of average density of a Scalar Core over the range of 4x4 to 20x20 LAB floorplans. (A Logic Array Block contains a column of 10 ALMs with local interconnect.) White squares indicate where the design did not fit in the floorplan.**

contain their critical paths, and thus improve performance. However, this approach gives the CAD tool a harder place-and-route problem to solve for otherwise identical hardware, and thus always gave worse results. Knowing this limitation, our new goal was to find a floorplan layout that would act as a hint to the CAD tool to enable the logic of a single tile to fit better into the structure of the FPGA hardware and improve performance, prior to any tiling. The Stratix FPGA architecture [17] places each logic resource type (ALMs, BRAMs, DSPs, etc...) into columns which span the height of the device and contain only one resource type each. Each cell in these columns has a relatively tall, narrow rectangular shape, and thus has more horizontal routing passing over it than vertical. We hypothesized that some floorplan shapes might take advantage of the cell aspect ratio and routing directional bias to improve logic usage and performance. We could then preserve the post-placement layout inside the floorplan, and simply "rubberstamp" the optimized tile across the FPGA.

Figure 2 shows a heatmap of the density of a Scalar Core when floorplanned into rectangular areas ranging from 4x4 to 20x20 LABs (Logic Array Blocks), with DSP and BRAM block excluded: The CAD tool may use DSP and BRAMs from outside the floorplan if necessary, and may move the floorplan anywhere on the FGPA to get good access to resources. A LAB contains a column of 10 ALMs with local interconnect. A Scalar Core requires over 64 LABs of logic area within the floorplan to successfully fit. The coordinates of a point on the heatmap denote a floorplan of the same rectangular shape, with the origin as the opposite corner. White squares indicate where the design did not fit in the floorplan.

As expected, forcing a tight fit (e.g.: less than 10x10) generally reduces F_{max}, and thus density, due to difficult routing. Any looser fit results in a middling density since the floorplan poses *fewer* constraints on placement and routing. Thus, even having a mostly full floorplan (or entire FPGA device) will not negatively affect density, regardless of geometry, but will also not improve it. A "sweet spot" at the 6x12 and 12x6 points, where the density jumps up by about 10% from about

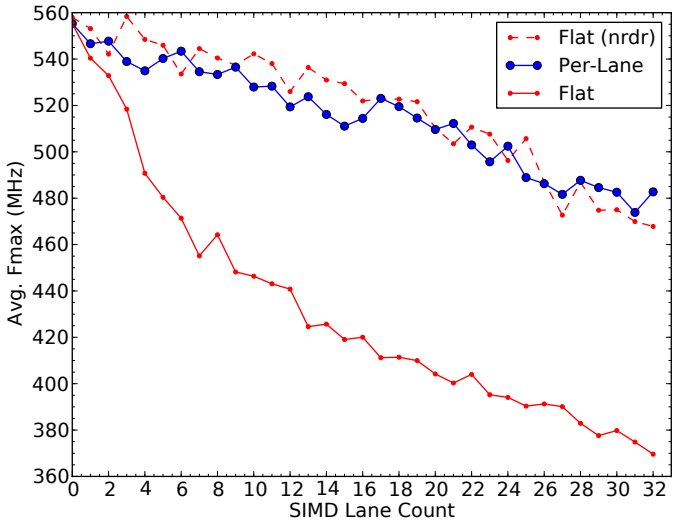


Fig. 3: **Average F_{max} of soft processor Cores with 0 to 32 SIMD lanes, partitioned as one unit ("Flat"), one unit but with no removal of duplicate registers ("nrdr"), and with the Scalar processor and each SIMD Lane placed in their own partitions ("Per-Lane"). The "Flat" F_{max} decreases due to excessive deduplication of multi-local logic. The "Per-Lane" and "nrdr" partitioning schemes preserve multi-local logic, thus a higher F_{max}.**

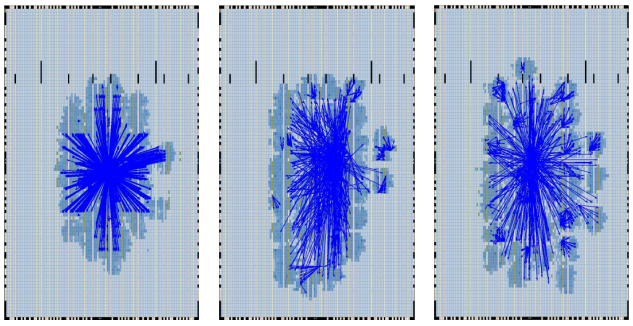
47 to almost 52 Peak MIPS per 100 eALMs, suggests that an optimum floorplan shape does exist: the 8x9 and 9x8 points enclose the same area, but have lower density.

However, if we compare the floorplanned density results with the density of the same unfloorplanned Scalar Core in the next section, we see that even an ample floorplan has a noticeable negative impact on density: with no floorplan constraint, the Scalar Core reaches a density of 54 Peak MIPS per 100 eALMs: 4% higher than the 6x12 and 12x6 sweet spots, and about 14% higher than most other floorplan geometries.

> Overall, floorplanning generally reduces density, regardless of shape and size, and provides no predictable benefit. Thus, barring project management concerns, a designer should tile an overlay without floorplanning, letting the CAD tool find a good placement solution itself.

IV. Partitioning Scheme Definitions

Throughout the remainder of this paper, we refer to various partitioning schemes with shorthand labels. In increasingly fine-grained order: "Flat", which places the entire design inside one partition; "nrdr", identical to "Flat" except with **n**o **r**emoval of **d**uplicate **r**egisters; "Per-Core", which places each soft processor into separate partitions; and "Per-Lane", which places each scalar processor and each of its SIMD lanes into their own separate partitions. We implement these schemes by declaring one or more modules as separate design partitions in the CAD tool. We do not consider schemes combining "nrdr" with "Per-Lane" or "Per-Core" as they always yielded worse results since not only would we prevent inter-module optimizations, we would also prevent any internal and usually beneficial optimizations.



(a) Flat (373 MHz) (b) nrdr (456 MHz) (c) Per-Lane (489 MHz)

Fig. 4: **Fanout of the source nodes of the top 100 critical paths for a "Flat", "nrdr", and "Per-Lane" partitioned 32-lane SIMD Core. The "Flat" critical paths (a) reach 373 MHz, and originate from 21 centrally placed nodes, resulting from deduplication optimizations, fanning out over the entire design. The "nrdr" critical paths (b) originate from 43 nodes, distributed (and duplicated) more evenly over the design, and reach 456 MHz. Finally, the "Per-Lane" critical paths (c) originate from 64 nodes, further spread out over the design, reaching 489 MHz.**

V. Partitioning a SIMD Core

Many overlays increase parallelism by tiling datapaths in a SIMD manner. We extended the Octavo soft processor [6] to support SIMD processing and observed the same logic in each SIMD lane consistently appearing in the worst critical paths, with their propagation delay increasing with the number of SIMD lanes. This multiply-local logic included the instruction distribution pipelines within each SIMD lane, as well as any common instruction decoding logic, such as the I/O address decoders. The CAD tool deduplicated all instances across all SIMD lanes, worsening the fanout distance of the remaining instance as the number of lanes increased. Using partitioning to preserve the multi-local logic avoids these artificial critical paths.

Our Scalar Core supports SIMD operations by duplicating its datapath once for each SIMD Lane and feeding instructions to all Lanes in lock-step, but lagging one instruction behind the Scalar Core (Figure 1(b)). Using fewer pipeline stages lowers F_{max}, while more pipeline stages do not significantly improve it. Each SIMD Lane contains its own copy of the instruction pipeline registers to scale them with the number of SIMD Lanes. We must explicitly duplicate the registers in the Verilog source itself since the CAD tool will not automatically do so to control longer paths caused by increasing fanout [12].

Figure 3 shows the average maximum operating frequency (F_{max}) of a soft processor Core with 0 to 32 SIMD lanes when partitioned as a whole unit ("Flat"), or same with no removal of duplicate registers ("nrdr"), and with the Scalar processor and each SIMD Lane placed in their own partitions ("Per-Lane"). Under "Flat", F_{max} decreases from 555 to 372 MHz due to excessive logic optimization: the CAD tool deduplicates equivalent local logic in each Lane down to a single instance which fans out to all SIMD Lanes, introducing critical paths which worsen as the number of Lanes increases. *Both the "nrdr" and "Per-Lane" partitioning schemes preserve the multiply-local logic in each Lane, thus limiting fanout distance and preserving F_{max} to up to 482 MHz, a 30% gain over "Flat".*

Figure 4 illustrates the preservation of multi-locality when we partition SIMD Lanes, or simply disable duplicate register removal, by spatially plotting on the FPGA device the fanout

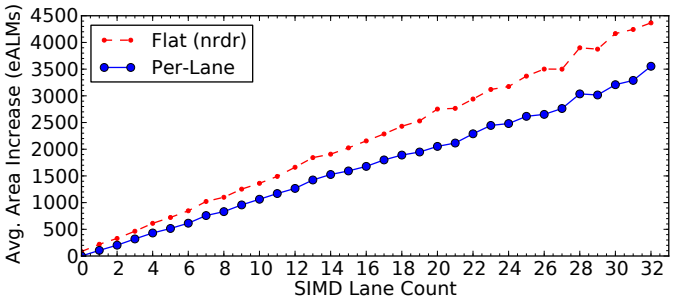


Fig. 5: **Average area increase (eALMs) of "Per-Lane" and "nrdr" partitioning schemes, for Cores with 0 to 32 SIMD Lanes, relative to the "Flat" area. For reference, the "Flat" area ranges linearly from 1,027 to 25,779 eALMs, over 0 to 32 SIMD Lanes.**

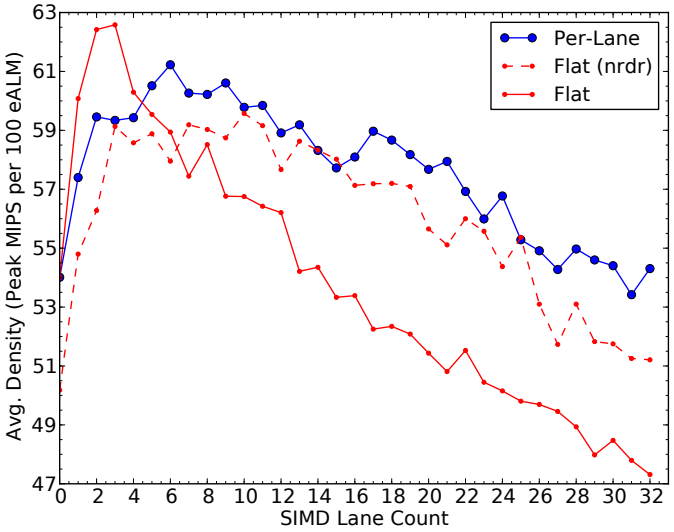


Fig. 6: **Average compute density (Peak MIPS per 100 eALMs) for "Per-Lane", "nrdr", and "Flat" partitioned Cores with 0 to 32 SIMD Lanes.**

of each source node in the top 100 critical paths of a 32-lane SIMD Core. The "Flat" critical paths (4(a)) reach 373 MHz and originate from 21 centrally placed nodes resulting from a deduplicated instruction pipeline fanning out over the entire design. The "nrdr" critical paths (4(b)) originate from 43 instruction pipeline nodes, distributed (and duplicated) more evenly over the design, and reach 456 MHz. Finally, the "Per-Lane" critical paths (4(c)) originate from 64 nodes (mostly I/O port address decoders) further spread out over the design, reaching 489 MHz because *we both preserved multi-locality across partitions and allowed optimizations within each partition.*

Figure 5 shows the increase of the average area (in eALMs) of Cores with 0 to 32 SIMD Lanes, under the "nrdr" and "Per-Lane" partitioning schemes, relative to the "Flat" area. While "nrdr" and "Per-Lane" use more area than "Flat" since they prevents logic deduplication, *the "nrdr" scheme does so indiscriminately, preventing optimizations which would normally occur inside "Per-Lane" partitions, resulting in a consistently larger total area.* For reference, the "Flat" area ranges linearly from 1,027 to 25,779 eALMs, over 0 to 32 SIMD Lanes.

Figure 6 combines the results of Figures 3 and 5 into a chart of the compute density, measured in Peak MIPS per 100 eALMs, of the "Per-Lane", "nrdr", and "Flat" partitioning schemes for Cores with 0 to 32 SIMD Lanes. Partitioning "Per-Lane" improves density once a design grows to the point of introducing new critical paths if multiply-local logic gets globally optimized into a single instance. Thus, the "Flat"

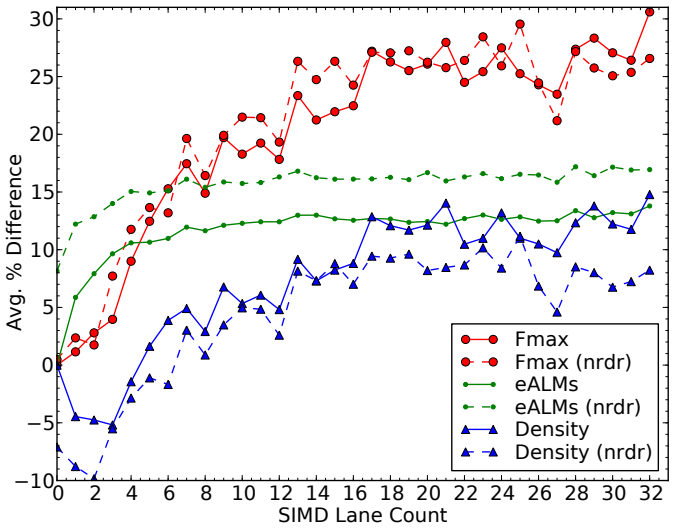


Fig. 7: **Average percent difference in F_{max}, area (eALMs), and compute density (Peak MIPS per 100 eALMs) of Cores with 0 to 32 SIMD Lanes, under "Per-Lane" and "nrdr" partitioning schemes, relative to "Flat" partitioning. The solid lines compare "Per-Lane" over "Flat", while the dashed lines compare "nrdr" over "Flat". A greater positive difference denotes a greater increase relative to "Flat".**

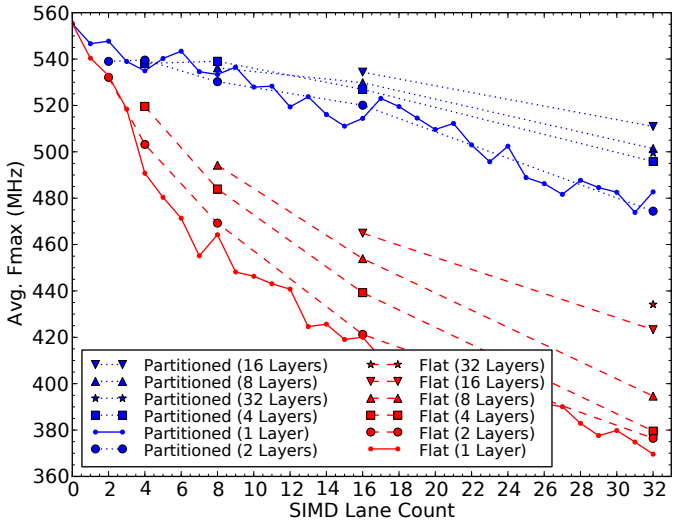


Fig. 8: **Average F_{max} (MHz) of "Flat" and "Per-Lane" partitioning of layered SIMD Cores, for 2, 4, 8, 16, and 32 layers, over the same total number of Lanes. The solid lines repeat the original single-layer SIMD results. The greatest increases happen at the limit cases where each layer contains a single SIMD Lane.**

scheme provides the best density with 4 SIMD Lanes or fewer (almost 63 at 3 Lanes, a roughly 5% increase over "Per-Lane" and "nrdr"), *while the multi-locality preservation of "Per-Lane" improves density for larger number of SIMD Lanes (approx. 54.5 at 32 Lanes, a roughly 14% increase over "Flat").* The "nrdr" scheme's consistently larger area reduces its density to below that of "Per-Lane", despite having similar F_{max}.

Figure 7 compares the results from Figures 3, 5, and 6 as the average percent differences in F_{max}, area (eALMs), and compute density (Peak MIPS per 100 eALMs) of Cores with 0 to 32 SIMD Lanes, under "Per-Lane" and "nrdr" partitioning schemes, relative to "Flat" partitioning. The solid lines compare "Per-Lane" over "Flat", while the dashed lines compare "nrdr" over "Flat". A greater positive difference denotes a greater increase relative to "Flat". "Per-Lane" and "nrdr" always increase F_{max}, by as much as 31%, with a corresponding 15% in compute density. *However, the consistently 5% larger area of "nrdr" lowers its density to below that of "Per-Lane", thus we do not consider "nrdr" in later experiments.* Finally, for 4 SIMD Lanes or fewer, "Flat" has 5 to 10% better density due to its greater logic optimizations and smaller fanout distances.

For SIMD Cores, partitioning "Per-lane" results in the greatest increase in F_{max} and density compared to keeping the entire Core within a single partition ("Flat"), even when also disabling removal of duplicate registers ("nrdr"). "Per-Lane" partitioning preserves the multi-locality within each SIMD Lane, avoiding the excessive fanout of instruction pipelines and I/O port address decoders otherwise optimized to a single instance by the CAD tool, while allowing beneficial local optimizations within each partition. The improvement from partitioning increases with the number of SIMD Lanes, reflecting the increasing multi-locality.

VI. Layering a SIMD Core

From Section V, we know that a SIMD Core implementation with fewer SIMD Lanes reaches a higher F_{max} due to lower instruction fanout, with or without deduplication of multi-local logic. Thus, we could divide N SIMD Lanes into M "layers" each containing N/M SIMD Lanes, with one Lane also acting as an instruction distribution pipeline stage to the next layer. Each successive layer lags the previous one by one instruction. Also, as the number of layers increases, we also effectively re-introduce the instruction distribution pipeline registers which the "Flat" variant optimizes away. However, these new registers sequentially depend on each other, thus the CAD tool cannot deduplicate them and increase their fanout. We explore all layering combinations for soft processor configurations with $[1, 2, 4, 8, 16, 32]$ SIMD lanes placed in the same increasing sequence of layers, up to N layers containing 1 SIMD Lane each.

Figure 8 shows the impact on the average F_{max} (MHz) of "Flat" and "Per-Lane" partitioning of layered SIMD Cores, for $[1, 2, 4, 8, 16, 32]$ layers, over the same total numbers of Lanes. *The greatest increases happen at the limit cases where each layer contains a single SIMD Lane.* For 32 lanes, the F_{max} of "Per-Lane" increases by only 30 MHz (+6%), compared to 65 MHz (+17%) for "Flat", which also benefits more throughout. Overall, "Per-Lane" partitioning of a single-layer SIMD Core always performs better than any layered "Flat" version, and without necessarily staggering execution across layers. Layering SIMD Lanes has little effect on total area (less than a 2% increase at most).

Sequentially pipelining SIMD Lanes into layers preserves their multi-locality in a way the CAD tool cannot optimize away, coarsely reproducing the effects of "Per-Lane" partitioning of each SIMD Lane. If a designer cannot use partitioning, then pipelining provides the next best alternative. However, partitioning by itself avoids the

TABLE I: Partitioned Meshes with 32 Datapaths

Shape (WxH)	SIMD Lanes	Datapaths (Total)	Scheme	Fmax (MHz)	Area (eALMs)	Density
Scalar Core						
1x1	0	1	Flat	555	1,027	54.0
4x8	0	32	Flat	362	36,379	31.8
4x8	0	32	Per-Core	481	36,651	42.0
1-Lane SIMD Core						
1x1	1	2	Flat	540	1,799	60.0
1x1	1	2	Per-Lane	547	1,904	57.5
4x4	1	32	Flat	380	32,226	37.7
4x4	1	32	Per-Core	438	32,718	42.8
4x4	1	32	Per-Lane	468	34,348	43.6
3-Lane SIMD Core						
1x1	3	4	Flat	518	3,313	62.5
1x1	3	4	Per-Lane	539	3,632	59.4
2x4	3	32	Flat	385	29,865	41.3
2x4	3	32	Per-Core	412	30,313	43.5
2x4	3	32	Per-Lane	461	32,321	45.6
31-Lane SIMD Core						
1x1	31	32	Flat	374	25,096	47.7
1x1	31	32	Per-Lane	473	28,385	53.3

TABLE II: Partitioned Meshes with 102 Datapaths

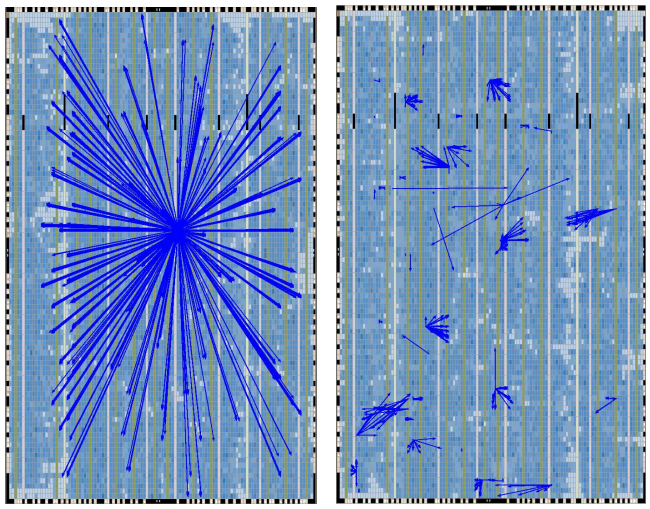
Shape (WxH)	SIMD Lanes	Datapaths (Total)	Scheme	Fmax (MHz)	Area (eALMs)	Density
1x1	0	1	Flat	555	1,027	54.0
17x6	0	102	Flat	287	115,655	25.3
17x6	0	102	Per-Core	434	115,245	38.4
6x17	0	102	Flat	284	114,803	25.2
6x17	0	102	Per-Core	437	115,775	38.5

programming complications of staggered execution and will always reach a higher F_{max}.

VII. Partitioning Meshes

To contrast with SIMD parallelism, we tile a number of Scalar and SIMD soft processor Cores into rectangular Meshes (Figure 1(a)) with horizontal and vertical point-to-point bidirectional links, and compare them to their building block Core in isolation. Tables I and II compare the F_{max} (MHz), area (eALMs), and density (Peak MIPS per 100 eALMs) of various Scalar and SIMD Cores tiled into rectangular Meshes of various shapes and sizes, under three increasingly fine-grained partitioning schemes: "Flat", which keeps the entire mesh in a single partition; "Per-Core", which places each Core (including its SIMD lanes, if any) into a partition; and "Per-Lane", which places each Scalar Core, and each SIMD Lane, into their own separate partitions. *For single-Core Meshes (1x1), "Per-Core" and "Flat" are equivalent, so we show "Flat" only.*

Table I compares Meshes with a total of 32 datapaths, along with their respective Core used as a building block, ranging from a 4x8 32-node Mesh of Scalar Cores, down to a 1x1 Mesh composed of a single 31-Lane SIMD Core. Under "Flat" partitioning, the dominating critical paths originate in the deduplication of a 3-bit free-running thread counter, found in the Control logic of each Core (see Figure 1), which does not depend on the contents of any memory or the output of any logic. Thus, the CAD tool considers them logically equivalent and optimizes them down to a single instance. The fanout from



(a) Flat (331 MHz) (b) Per-Lane (489 MHz)

Fig. 9: **Fanout of the source nodes of the top 100 critical paths for "Flat" and "Per-Lane" partitions of a 102-core 17x6 tiled mesh of Scalar Cores. The "Flat" critical paths reach 331 MHz and originate from only 2 centrally placed nodes. In contrast, the "Per-Lane" critical paths originate from 61 unrelated and dispersed nodes, reaching an F_{max} of 489 MHz.**

this single counter instance causes a drop in F_{max} proportional to the number of Mesh nodes. *We can demonstrate this drop by simply separating the Cores into their own partitions ("Per-Core") to preserve the multi-locality of the thread counter, with F_{max} improving in proportion to the number of Mesh nodes:* a 4x8 Mesh of Scalar Cores improves its F_{max} by 33%, while the equivalent but half-size 4x4 Mesh of 1-Lane SIMD Cores see a proportional improvement of 15%, and the quarter-size equivalent 2x4 Mesh of 3-Lane SIMD Cores see a 7% improvement. Note that all these Meshes have similar total areas, so the gain from "Per-Core" partitioning only depends on the number of Cores, not their individual area.

Even after "Per-Core" partitioning, Mesh nodes with SIMD Lanes still have to preserve the multi-locality of their instruction pipeline and I/O port address decoders. Partitioning "Per-Lane" suffices, since this scheme places each Scalar component of each SIMD Core into its own partition, and thus also preserves the multi-locality of the thread counter. In each case shown in Table I, when going from "Per-Core" to "Per-Lane" partitioning, the F_{max} increases with the number of datapaths in each Core: the 4x4 Mesh of 1-Lane Cores improves by 7%, the 2x4 Mesh of 3-Lane Cores improves by 12%, and the 1x1 Mesh of a single 31-Lane SIMD Core improves by 26%. *Once we preserve all the multi-localities, the final F_{max} of all the different 32-datapath Meshes lie within a 4% range.*

Table II shows two shapes (17x6 and 6x17) of a Mesh of 102 Scalar Cores, the largest number possible before running out of M9K BRAMs[3]. Spanning the entire FPGA device (see Figure 9), this 102-node Mesh uses 67% of all logic, broken down by Quartus as: 76% of all memory bits, 74% of all flip-flops, and 24% of all ALUTs. The aspect ratio of the 102-core Mesh has negligible effect on speed and area, and partitioning "Per-Core" improves F_{max} by 51 to 54%, again showing the preservation of the multi-locality of the thread counter and that our gains do not depend on being able to spatially separate each datapath, as Figure 4 might imply.

[3]Each Scalar Core uses 12 M9K BRAMs, leaving 11 unused out of 1235.

Figure 9 illustrates the fanout of the source nodes of the top 100 critical paths for "Flat" and "Per-Lane" partitions of a 102-core 17x6 tiled mesh of Scalar Cores. The "Flat" critical paths reach 331 MHz, and despite having no shared instruction pipelines or address decoders, originate from only **2** centrally placed nodes resulting from the deduplication of free-running thread counters common to the Control logic in all Cores (see Figure 1), now fanning out over the entire design. In contrast, the "Per-Lane" critical paths originate from 61 unrelated and dispersed nodes, and reach an F_{max} of 489 MHz.

Since partitioning prevents deduplication of multi-local logic, it increase the area of a tiled design. *However, the area increases only proportionally to the size of the preserved multi-local logic, not from partitioning itself.* For example, the area of the 102-core 6x17 Mesh increases by less than 1% under "Per-Lane" partitioning, reflecting the tiny area of the preserved multi-local 3-bit counters. At the other extreme, the area of the 1x1 Mesh with 32 datapaths increases by 13% due to the larger area of multi-local SIMD instruction pipelines and address decoders. In the middle, the area of the 4x8, 4x4, and 2x4 Meshes (also with 32 datapaths) increases by 8% or less since the multi-localities include both the smaller counters and the larger instruction pipelines and address decoders. *In all cases, no relation exists between the increase in F_{max} and the increase in area.*

The changes in F_{max} and area as we scale-up Meshes suggest that: (i) we may be getting near-optimal speed preservation from the CAD tool when partitioning; (ii) the CAD tool introduces some unknown area overhead, causing a drop in density, even though we added no extra hardware when tiling. Under "Per-Lane" partitioning, tiling a Scalar Core 32 times in the 4x8 Mesh decreases F_{max} only 13% (with similar decreases for the 4x4 and 2x4 Meshes). Increasing the tiling 3.18x further to 102 cores decreases F_{max} by only an additional 9%, down to 78% of the original speed of a single Scalar Core. *A 22% drop in speed over 102x scaling supports our original intuition that multiple concurrent tiles should run at similar speeds to a single tile.*

Intuitively, we expect that tiling without adding any other hardware should scale the total area by the number of tiles. However, when tiling a Scalar Core 32 times (4x8 Mesh) the area increases 35.7x, and 112.2x when tiling it 102 times: a 10 to 11% overhead. If the area increased as expected, and assuming the same final F_{max}, the density of the 102-core mesh would reach 42.3, placing it within 8% of the densest 2x4 Mesh (or within 11% of a similarly speculative 4x8 Mesh). *Thus, we speculate that some area overhead introduced by the CAD tool, rather than the decrease in F_{max}, primarily causes the drop in density as Meshes scale up.*

Partitioning has no significant effect on CAD time. For example, for the 17x6 102-core mesh, the total CAD time goes from 2h:28m when "Flat", to 2h:17m (-7%) when partitioned "Per-Lane". Synthesis and Place-and-Route consume most of the total time. The synthesis time went from 0h:18m to 0h:12m (-33%), since the CAD tool can synthesize each partition in parallel, but still represents only 12 to 8% of the total time. The Place-and-Route time went from 2h:05m to 1h:55m (-8%), but still represents 84% of the total, with or without partitioning. Furthermore, once partitioned for performance, nothing prevents a designer from using partitioning (and optionally, floorplanning) to enable incremental re-compilation and reduce future CAD time by containing the effect of design changes [13]–[15].

> Different kinds of tiling (SIMD Lanes vs. Mesh) expose different multi-localities in the same tile hardware, which we preserve with different partitioning schemes to recover lost performance. Partitioning itself has no area impact: any increase in area purely depends on the area of the preserved multi-local logic, with no correlation to the amount of lost/regained performance. The CAD tool preserves the operating frequency of partitioned tiles extremely well as their number increases, but introduces an unexplained proportional area overhead, independent of partitioning, which may explain the decreasing compute density with scaling. Lastly, partitioning has no significant effect on total CAD time, even without incremental recompilation.

VIII. Related Work

Past work on partitioning and FPGAs address a different, but related problem: how to partition the netlist of an ASIC project across multiple FPGAs for simulation [18]. Fang and Wu [19], [20] found that using the design hierarchy to guide partitioning would lead to higher logic block utilization and lower I/O pin utilization, which commonly formed the bottleneck when partitioning over multiple FPGAs, and resembles our avoidance of high-fanout paths. Some early work by Vahid, Frank, Le, and Hsu [21], [22] pointed to the advantages of functional partitioning, the kind we do in this paper, over structural partitioning, where partitioning occurs after the synthesis of a final, flattened netlist. They observed a similar control of the critical paths (and area increase) by avoiding sharing logic between functions (i.e.: "multi-local" logic).

IX. Conclusions

Through our study of high-speed tiled FPGA overlays, we found the notion of "multi-local" (repeated and logically-equivalent across tiles) logic useful to describe the origin of the worst critical paths which emerge when tiling datapaths or even entire processors: By default, the CAD tool performs redundancy elimination ("deduplication") on all multi-local logic, reducing it to a single instance which then fans out to all tiles, introducing large and unnecessary critical paths for little area savings.

We found that partitioning the major datapaths of a high-speed tiled system into separate netlists suffices to prevent harmful global deduplication of multi-local logic, while preserving beneficial local optimizations. Partitioning provides: (i) a simpler approach than the existing per-node micro-management of a design description or of the CAD tool settings; (ii) a lower area increase than simply disabling duplicate register removal in the CAD tool, only depending on the area of preserved multi-local logic; (iii) greater performance benefits than forcibly introducing sequential dependencies via pipelining; (iv) a performance increase that scales with the number of tiles; (v) improved performance with no significant change in total CAD time.

X. Further Work

Although manual partitioning of tiled overlays requires relatively little work from the designer, we believe the CAD tool might automatically detect multi-local logic and take some action other than complete redundancy elimination. (i) The CAD tool could detect repeated deduplication of the same logic across multiple modules and declare that logic multi-local, optionally alerting the designer. (ii) The CAD tool could "restart" the optimization of multi-local logic after a certain number of deduplications, ensuring that sufficient copies remain to avoid large critical paths, while still reducing area. (iii) The CAD tool could use the location of the multi-local logic to automatically partition its enclosing modules, automating the approach we used throughout this paper. Some prior work by Dehkordi, Brown, and Borer has explored this kind of automated modular partitioning, but only to control incremental recompilation time, with no impact on frequency [23].

Since partitioning can increase the operating frequency more than it does logic area, the power density and power consumption of the system may also increase. Alternately, the power consumption may also decrease due to driving fewer device-wide critical paths when partitioned. For designs under power constraints, the CAD tool could explore the dynamic and static power trade-offs of full or partial partitioning.

Finally, although the CAD tool preserves the performance of partitioned tiles very well as their number increases, it seems to add an unexplained and consistent area overhead when tiling, causing a drop in compute density. Future CAD tools should limit this overhead to better enable large-scale tiled overlays.

Acknowledgements

The authors gratefully acknowledge funding from the Queen Elizabeth II World Telecommunication Congress Graduate Scholarship in Science and Technology, the Walter C. Sumner Foundation, Altera, and NSERC. Computations, *totalling over 24 CPU-years*, were performed on the GPC supercomputer at the SciNet HPC Consortium [24]. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto. The authors also thank Altera for providing sufficient Quartus licenses to support our research.

References

[1] P. Yiannacouras, J. G. Steffan, and J. Rose, "VESPA: Portable, Scalable, and Flexible FPGA-Based Vector Processors," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 2008, pp. 61–70.

[2] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, "VEGAS: soft vector processor with scratchpad memory," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2011, pp. 15–24.

[3] A. Severance and G. Lemieux, "VENICE: A compact vector processor for FPGA applications," *IEEE International Conference on Field Programmable Technology (FPT)*, pp. 261–268, Apr. 2012.

[4] H. Y. Cheah, S. A. Fahmy, D. L. Maskell, and C. Kulkarni, "A Lean FPGA Soft Processor Built Using a DSP Block," in *Proceedings of the ACM/SIGDA international Symposium on Field Programmable Gate Arrays (FPGA)*, 2012, pp. 237–240.

[5] H. Y. Cheah, S. Fahmy, and D. L. Maskel, "iDEA: A DSP block based FPGA soft processor," *IEEE International Conference on Field Programmable Technology (FPT)*, pp. 151–158, 2012.

[6] C. E. LaForest and J. G. Steffan, "OCTAVO: An FPGA-Centric Processor Family," in *Proceedings of the 20th ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA)*, Feb. 2012, pp. 219–228.

[7] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core CPU accelerator," *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 222–232, 2008.

[8] M. Labrecque, P. Yiannacouras, and J. G. Steffan, "Scaling Soft Processor Systems," *16th International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 195–205, Apr. 2008.

[9] D. Unnikrishnan, J. Zhao, and R. Tessier, "Application Specific Customization and Scalability of Soft Multiprocessors," *17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pp. 123–130, Apr. 2009.

[10] F. Anjam, M. Nadeem, and S. Wong, "A VLIW Softcore Processor with Dynamically Adjustable Issue-slots," *International Conference on Field-Programmable Technology (FPT)*, pp. 393–398, 2010.

[11] W. Zhang, V. Betz, and J. Rose, "Portable and Scalable FPGA-Based Acceleration of a Direct Linear System Solver," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 1, pp. 1–26, Mar. 2012.

[12] R. Scoville, "Register Duplication for Timing Closure," Altera, Tech. Rep., 2011. [Online]. Available: http://www.alterawiki.com/uploads/a/aa/Register_Duplication_for_Timing_Closure.pdf

[13] C. Zeh, "Incremental Design Reuse with Partitions," *Xilinx XAPP918 (v1.0)*, vol. 918, pp. 1–17, 2007. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp918.pdf

[14] D. Chen and D. Singh, "Line-level incremental resynthesis techniques for FPGAs," *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 133–142, 2011.

[15] Altera, "Best Practices for Incremental Compilation Partitions and Floorplan Assignments," *Quartus II Handbook Version 13.0*, vol. 1, pp. 1–54, November 2012. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii51017.pdf

[16] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 5–14, 2011.

[17] D. Lewis, V. Betz, D. Jefferson, and A. Lee, "The Stratix Routing and Logic Architecture," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 12–20, 2003.

[18] D. Behrens, K. Harbich, and E. Barke, "Hierarchical Partitioning," *International Conference on Computer Aided Design (ICCAD)*, pp. 470–477, 1996.

[19] W.-J. Fang and A. C.-H. Wu, "Performance-driven multi-FPGA partitioning using functional clustering and replication," *Proceedings of the 35th IEEE/ACM Design Automation Conference (DAC)*, pp. 283–286, 1998.

[20] ——, "Multiway FPGA partitioning by fully exploiting design hierarchy," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 1, pp. 34–50, Jan. 2000.

[21] F. Vahid, T. D. Le, and Y.-C. Hsu, "A comparison of functional and structural partitioning," *9th International Symposium on System Synthesis (ISSS)*, pp. 121–126, 1996.

[22] ——, "Functional partitioning improvements over structural partitioning for packaging constraints and synthesis: tool performance," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 3, no. 2, pp. 181–208, Apr. 1998.

[23] M. E. Dehkordi, S. D. Brown, and T. P. Borer, "Modular Partitioning for Incremental Compilation," *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–6, 2006.

[24] C. Loken, D. Gruner, L. Groer, R. Peltier, N. Bunn, M. Craig, T. Henriques, J. Dempsey, C.-H. Yu, J. Chen, L. J. Dursi, J. Chong, S. Northrup, J. Pinto, N. Knecht, and R. V. Zon, "SciNet: Lessons Learned from Building a Power-efficient Top-20 System and Data Centre," *Journal of Physics: Conference Series*, vol. 256, p. 012026, Nov. 2010.