Unified Incremental Physical-Level and High-Level Synthesis

Zhenyu (Peter) Gu, *Student Member, IEEE,* Jia Wang,
Robert P. Dick, *Member, IEEE,* Hai Zhou, *Senior Member, IEEE*

***Abstract*— Achieving design closure is one of the biggest challenges for modern VLSI designers. This problem is exacerbated by the lack of high-level design automation tools that consider the increasingly important impact of physical features, such as interconnect, on integrated circuit area, performance, and power consumption. Using physical information to guide decisions in the behavioral-level stage of system design is essential to solve this problem. In this paper, we present an incremental floorplanning high-level synthesis system. This system integrates high-level and physical design algorithms to concurrently improve a design's schedule, resource binding, and floorplan, thereby allowing the incremental exploration of the combined behavioral-level and physical-level design space. Compared with previous approaches that repeatedly call loosely coupled floorplanners for physical estimation, this approach has the benefits of efficiency, stability, and better quality of results. The average CPU time speedup resulting from unifying incremental physical-level and high-level synthesis was 24.72$\times$ and area improvement was 13.76%. The low power consumption of a state-of-the-art, low-power, interconnect-aware high-level synthesis algorithm was maintained. The benefits of concurrent behavioral-level and physical design optimization increased for larger problem instances.**

I. Introduction

Process scaling has enabled the production of integrated circuits (ICs) with millions of transistors. This has allowed the design of more full-featured and high-performance ICs. However, these increased capabilities have come at a cost. In order to deal with increased design complexity and size, it is becoming increasingly important to automate the higher levels of the design process.

High-level synthesis systems [1]–[4] automatically convert behavioral, algorithmic, descriptions of design requirements, e.g., control data flow graphs (CDFG) [5], into optimized register-transfer level (RTL) descriptions in languages such as VHDL or Verilog. Based on a behavioral description, a high-level synthesis system determines an allocation of resources, assignment of operations to resources, and a schedule for operations, in an attempt to satisfy the design specifications and minimize some combination of delay, area, and power consumption [6]–[17]. Recently, in order to improve design area or performance estimation, a number of researchers have considered the impact of physical details, e.g., floorplanning information, on high-level synthesis [18]–[23].

In the past, it was possible for high-level synthesis algorithms to focus on logic, i.e., functional units such as adders and multipliers. The contribution of wire delay and area was typically neglected without much loss of accuracy. Focusing on logic was once reasonable since logic was responsible for the majority of delay and power consumption. However, process scaling into the deep sub-micron realm has changed the focus of VLSI design from transistors to global interconnect. It is no longer possible to simplify the high-level synthesis problem by ignoring interconnect.

Taking interconnect cost into consideration during high-level synthesis has attracted significant attention. In previous work [24]–[28], the number of interconnects or multiplexers was used to estimate interconnect cost. The performance and power impacts of interconnect and interconnect buffers are now first-order timing and power considerations in VLSI design [29]. It is no longer possible to accurately predict the power consumption and performance of a design without first knowing enough about its floorplan to predict the structure of its interconnect. This change has dramatically complicated both design and synthesis. For this reason, a number of researchers have worked on interconnect-aware high-level synthesis algorithms [30]–[32]. These approaches typically use a loosely coupled independent floorplanner for physical estimation. Although this technique improved on previous work by allowing estimation of physical properties, there are two drawbacks for this approach. First, the independent floorplanner may not be stable, i.e., a small change in the input netlist may result in a totally different floorplan. A *move* is a discrete change made to a solution during optimization that results in transition to a new position in the solution space. Floorplan instability may result in a high-level synthesis algorithm that bases its moves on cost functions without continuity. Second, even if a floorplanner is stable, creating a floorplan from scratch for each high-level synthesis move is inefficient, given the fact that the new floorplan frequently has only small differences with the previous one. The constructive approach works for small problem instances but is unlikely to scale to large designs. New techniques for tightly coupling behavioral and physical synthesis that dramatically improve their combined performance and quality are now necessary.

Incremental automated design promises to build tighter relationship between high-level synthesis and physical design, improving the quality of each [33]–[35]. A number of high-

This work is supported by the NSF in part under award CCR-0347941 and in part under award CCR-0238484.

The authors are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 USA (e-mail: zgu646@eecs.northwestern.edu, jwa112@eecs.northwestern.edu, dickrp@eecs.northwestern.edu, haizhou@eecs.northwestern.edu)

level synthesis algorithms are based on incremental optimization and are, therefore, amenable to integration with incremental physical design algorithms. This has the potential of improving both quality and performance. Incremental methods improve quality of results by maintaining important physical-level properties across consecutive physical estimations during synthesis. Moreover, they shorten CPU time by reusing and building upon previous high-quality physical design solutions that required a huge amount of effort to produce.

This paper describes an incremental high-level synthesis system that reduces synthesis time dramatically while producing ICs with better area and low power consumption compared to a state-of-the-art power-aware high-level synthesis algorithm. The benefits of this approach increase with increasing problem size and complexity. Our work is based on the interconnect-aware high-level synthesis tool, ISCALP [31], which was based on the low-power datapath synthesis tool, SCALP [15]. We reuse the power modeling and iterative-improvement high-level synthesis framework from ISCALP. However, this work differs from previous work in that a truly incremental floorplanner is used to estimate the interconnect structure [36], instead of a fast constructive algorithm. Moreover, the high-level synthesis algorithm, itself, is made incremental. As shown in Section V, this resulted in an average speedup of $24.72\times$ and an average area improvement of 13.76%, while maintaining the low power consumption of a state-of-the-art power-aware high-level synthesis algorithm. In addition, wire delay is considered in this work to guarantee that the implementation meets its performance requirements.

This paper is organized as follows. Section II explains the motivation for this work. Section III describes the design flow for the proposed high-level synthesis system. The details of our incremental floorplanner are introduced in Section IV. Experimental results are presented in Section V. Conclusions and future work are presented in Section VI.

II. Motivation

In this section, we first present definitions useful in the discussion of high-level synthesis. Then motivational examples are given based on our observations of the synthesis process. Examples are used to explain and motivate the use of unified high-level and physical-level optimization.

A. *Definition*

The input to ISCALP is a control-data flow graph (CDFG), G, an input arrival (and output sampling) period, T_s, and a library, L, of functional units for data path implementation. ISCALP produces an RTL circuit in which power consumption (including logic and wire power consumption) and estimated area are optimized. The ISCALP algorithm has two loops. Given the supply voltage, the outer loop incrementally reduces the number of control steps, $csteps$, from its maximum to minimum value, where $csteps$ is defined as

$$csteps = T_s \times f \tag{1}$$

or alternatively,

$$T_{clock} = T_s / csteps \tag{2}$$

In the above equations, each control step corresponds to a time period of one clock cycle, and the sample period T_s is the constraint on the input data rate. The solution got from high-level synthesis must be able to process an input sample before the next one arrives. For a given specification, the sample period is fixed. Hence, $csteps$ indicates the number of clock cycles required to process an input sample. The variable f is the chip clock frequency. T_{clock} is the chip clock period. ISCALP searches for the lowest-power architecture for each possible value of $csteps$.

Given a value of $csteps$, which allows the clock period to be determined, the inner loop first uses the fastest available functional unit from the library to implement each operation. An as-soon-as-possible (ASAP) schedule is then generated for the initial solution to determine whether it meets its timing requirements. The initial solution is then further optimized. Having obtained an initial solution that meets the sample period constraint for the current value of $csteps$, the iterative improvement phase attempts to improve the architecture by reducing the switched capacitance while satisfying the sample period constraints. More details can be found in literature [15], [31] .

B. *Motivating Example*

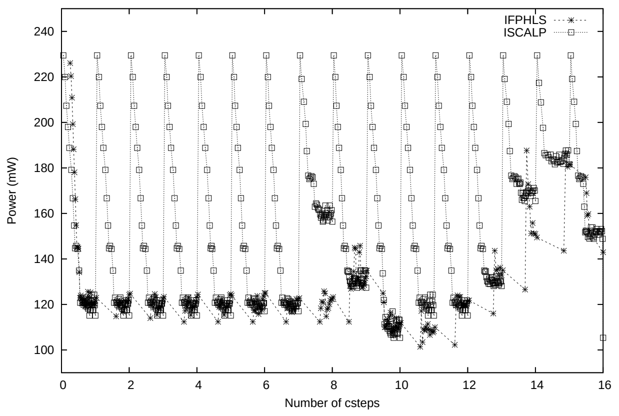
ISCALP employs a fast constructive slicing floorplanner based on netlist partitioning and rotation/orientation selection to obtain a floorplan optimized for wire length and area [37], [38]. Although it improved on its predecessors by considering the impact of floorplanning on synthesis, there are several drawbacks to this approach.

First, an incremental high-level synthesis algorithm only changes a small portion of the modules and connections in each move. However, a constructive (conventional) floorplanner always starts floorplanning from scratch. It is not efficient because it does not reuse the floorplan information obtained during the previous run. Moreover, it is possible that the newly-produced floorplan will be totally different from the previous one, despite only small changes in the set of modules and interconnections. This lack of autocorrelation in floorplan solutions may result in a high-level synthesis algorithm basing its future moves on information that immediately becomes invalid after the moves are applied.

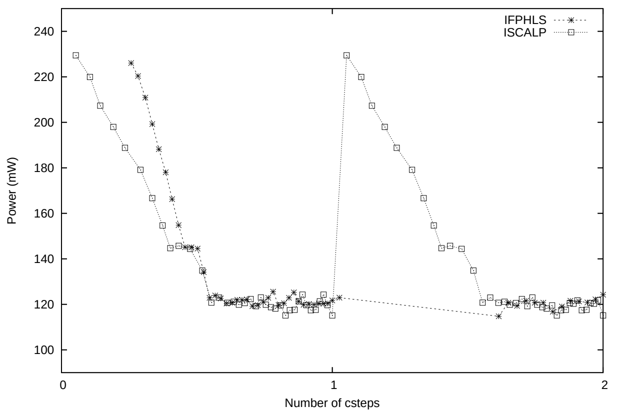
Second, the efficiency of the constructive slicing structure floorplanner decreases dramatically for blocks with non-unity aspect ratios (ISCALP assume blocks with unity aspect ratio). As a result of constraining the solution space to slicing floorplans, it is prone to reaching sub-optimal solutions. However, simply replacing the slicing floorplanner with a high-quality floorplanner would result in unacceptably-high CPU time.

To solve these problems, we propose an incremental iterative improvement high-level synthesis algorithm tightly integrated with a high-quality incremental floorplanner. This synthesis system is called IFP-HLS, i.e., incremental floorplanning high-level synthesis. We run the same benchmarks on both ISCALP and IFP-HLS, listing the number of merge operations and CPU time for each benchmark in Table I.

$$t_{total} = N_{moves} * (t_{HLS} + t_{fp}) \tag{3}$$



(a) Power consumption of intermediate solution during optimization for all the values of csteps.



(b) Power consumption of intermediate solution during optimization for the values of csteps from 0 to 2.

Fig. 1. Power consumption of intermediate solution during optimization of ISCALP & IFP-HLS.

As Equation 3 shows, the CPU time of the high-level synthesis run can be divided into two parts: high-level synthesis moves and the resulting physical design carried out by the floorplanner. As shown in Table II, floorplanning is the most time consuming of these. It uses at least 75.69% of the CPU time on average for both ISCALP and IFP-HLS. As shown in Table I, IFP-HLS achieves an average reduction of 50% in the number of high-level synthesis merge operations compared to ISCALP. This results in a large reduction in floorplanner CPU time. The reduction in moves, and CPU time, is mainly due to the incremental high-level synthesis and floorplanning algorithms used in IFP-HLS. Many high-level synthesis moves result in time-consuming changes to the floorplan. IFP-HLS can greatly reduce CPU time by reducing the number of merge operations, especially for larger benchmarks which have bigger solution space to explore.

Figure 1 illustrates the power consumptions of intermediate solutions during optimization in ISCALP and IFP-HLS. For each value of csteps, we plot the intermediate solutions produced by the optimization algorithm. Note that these intermediate solutions all have the same value of csteps. Incremental optimization allows IFP-HLS to focus on the most promising (low-power) regions of the solution space while ISCALP must essentially restart optimization for each new potential clock frequency. This allows improvement to both optimization time and solution quality for IFP-HLS.

Note that ISCALP starts the floorplanner from scratch after each high-level design change. The incremental physical and architectural optimization used in IFP-HLS reduces CPU time dramatically, especially for large applications. Table I indicates that the average CPU time speedup is 24.72×. The improvement is greatest for the largest benchmarks. For example, when run on Random300, ISCALP does not finish within 5 days, while IFP-HLS finishes within 4 hours. In addition, IFP-HLS achieves 13.76% improvement in area compared to ISCALP.

The above examples clearly illustrate the value of using unified incremental physical-level and high-level design synthesis. As shown in detail in Section V, this approach improves both design quality and CPU time.

III. Incremental High-Level Synthesis

In this section, we describe our incremental floorplanning high-level synthesis algorithm (IFP-HLS). IFP-HLS is built upon ISCALP [31]. However, incorporating incremental floorplanning required substantial changes to that algorithm, resulting in a new low-power incremental floorplanning high-level synthesis algorithm. IFP-HLS considers both datapath and interconnect power consumption. As shown in Figure 2, the CDFG is simulated with typical trace in order to profile the switching activity of each operation and data transfer edge. A RTL design library is set up to provide the power and area information. The profiling information combined with this RTL design library and floorplanner is then used to evaluate the power consumption of both datapath and interconnect. IFP-HLS uses a new incremental method for improving functional unit binding during high-level synthesis. Although this improvement, alone, would result in a reduction in synthesis time, its motivation was to facilitate the integration of an incremental floorplanning algorithm with high-level synthesis in order to improve solution quality and reduce synthesis time. This allows the high-level synthesis algorithm to determine the physical position of each module during optimization, enabling interconnect power consumption and delay estimation.

A. Incremental High-level Synthesis Framework

In this section, we describe our incremental high-level synthesis tool, IFP-HLS. The flowchart of IFP-HLS is shown in Figure 2. IFP-HLS differs from ISCALP in a number of ways. Instead of generating an initial solution for each value of $csteps$, IFP-HLS only generates one solution at the maximum value of $csteps$ and incrementally changes the solution as $csteps$ decreases. Thus, in addition to using incremental floorplanning, IFP-HLS also eliminates redundant moves by taking advantages of incremental steps in high-level synthesis. Initially, we still use an ASAP schedule and fully parallel allocation to estimate whether there exists a valid solution for the current value of $csteps$. If not, it is not necessary to do further moves for the current number of control steps because a binding that further reduces the finish

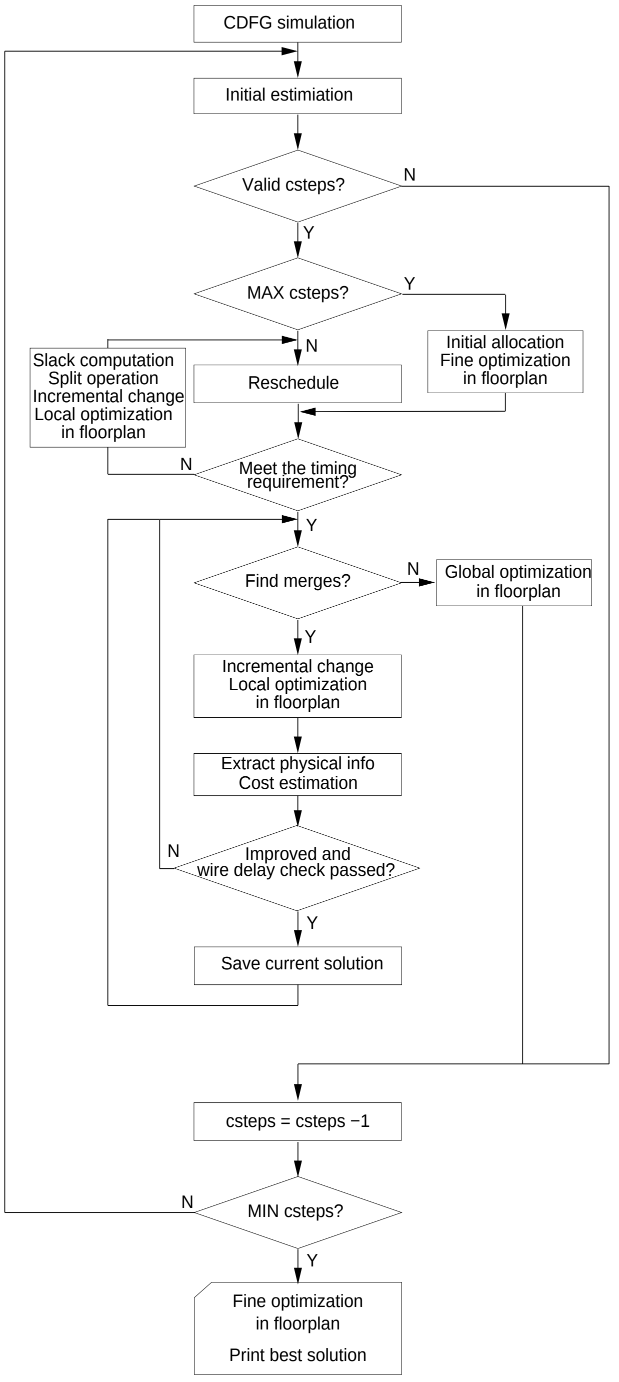


Fig. 2. Incremental high-level synthesis algorithm.

time of an ASAP schedule is not possible. However, if an ASAP schedule meeting the timing requirements is possible, we will use the best solution from the previous value of $csteps$ and reschedule it based on the current value, which is equal to the previous $csteps$ minus 1. If, after rescheduling, the solution meets its timing requirements, rebinding is not necessary. Otherwise, it will be necessary to parallelize some of the operations to improve performance. The *split move* is used to eliminate resource contention by splitting a pair of operations that were initially assigned to the same functional unit onto separate functional units. A detailed description of the split move may be found in Section III-B.

For a given value of $csteps$, when a move is chosen, IFP-HLS incrementally changes the floorplan to see whether the change improves solution quality. If so, the change is accepted. Otherwise, the change is rejected and other moves are attempted. This technique differs from that in ISCALP. In ISCALP, floorplanning is only done at the end of each $csteps$ iteration; it does not take advantage of solution correlation to save effort across $csteps$ values. ISCALP uses only power consumption to guide high-level synthesis moves. In contrast, IFP-HLS uses a weighted sum of area and power consumption (pW), with a ratio of $1\,\mu\mathrm{m}^2$ to $5\,\mathrm{pW}$, in order to evaluate solution quality.

A high-quality incremental floorplanner was developed and incorporated into IFP-HLS to guide high-level synthesis moves. Each time the high-level synthesis algorithm needs physical information, it extracts that information from the current, incrementally generated, floorplan. Costs derived from the floorplan are also used to guide high-level synthesis moves. By using incremental floorplanning, closer interaction between high-level synthesis and physical design is possible, i.e., the high-level synthesis algorithm may determine the impact of potential changes to binding upon physical attributes such as interconnect power consumption and area.

The core idea of incremental design is to maintain good physical-level properties across consecutive physical estimations during high-level synthesis moves. It is possible to apply the idea of using an incremental optimization framework to integrate other algorithms, provided that the algorithms at each level of design can be made incremental. Let us consider a few other examples. In force-directed scheduling, all the operations may be scheduled iteratively in the order of increasingly cost and the cost of scheduling each unscheduled operation is updated after each operation is scheduled. This provides a potential opportunity to tightly integrate an incremental floorplanner to physical information feedback. For maximal clique based resource sharing, since it is a NP-hard problem, a heuristic algorithm will be used in practice. As long as the heuristic algorithm itself is iterative, it can be made incremental.

In summary, IFP-HLS performs scheduling, allocation, and binding by iteratively changing $csteps$ and determining whether operations need to be rescheduled or re-bound (split) in order to meet timing constraints. At each step the floorplan is updated and re-optimized.

B. Extended Move

This subsection describes the split moves, rescheduling, and a new graph technique to determine split locations.

We observed that when $csteps$ decreases by one, each individual operation takes, at most, the same number of control steps as it did for the previous value of $csteps$. Given that $csteps$ is no less than the previous $csteps$ minus one, we can conclude that the ASAP schedule for the previous value of $csteps$ violates the deadline for the current value of $csteps$ by, at most, one clock cycle. We will use node i's slack, S_i, to represent this information, which is defined as follows:

$$S_i = LST_i - EST_i \tag{4}$$

Here, EST_i is the earliest start time and LST_i is the latest start time which were computed by a topological sort. Hardware resource contention has already been considered.

Nodes with non-negative slack values do not imply timing violations. However, nodes with slack values of -1 cause timing violations, i.e., they must be executed one cycle earlier. These timing violations can be removed by splitting merged operations which, although useful for previous values of $csteps$, now harm performance. Based on this observation, the split move is used to eliminate timing violations. Therefore, the whole high-level synthesis algorithm is implemented in an incremental way from maximum to minimum values of $csteps$ without rebinding from scratch at each value of $csteps$. Few changes to binding and scheduling are required as a result of single-unit change to $csteps$. However, in order to meet timing requirements, it is sometimes necessary to split operators mapped to the same functional unit. The split move makes it possible to quickly apply these isolated changes. Previous high-level synthesis systems, e.g., SCALP and ISCALP, started from a fully parallel implementation for each value of $csteps$ and repeatedly merged operators to reduce area. Although both techniques are reasonable in the absence of an integrated floorplanner, the incremental approach used in IFP-HLS speeds optimization (without degrading solution quality) by requiring far fewer changes to the floorplan. Used together, the split and merge moves allow complete exploration of the solution space. However, the primary goal of changing the number of control steps is meeting timing constraints. We therefore start our exploration of the solution space at the most promising region by iteratively splitting functional units on the critical timing path.

Algorithm 1 Reschedule and Split Procedure

```
1: Reschedule the design
2: Compute slacks of all operations
3: **while** there exists negative slack **do**
4:     Compute slack of all operations
5:     Construct graph including all the operations and edges
       with negative slack
6:     Use maximum flow to find the minimum cut in the
       graph
7:     Do the split move
8:     Reschedule the design
9: **end while**
```

The reschedule and split procedure is shown in Algorithm 1. We will give an example to further describe this procedure. Consider the data flow graph shown in Figure 3(a), in which arrows represent the data dependencies. Scheduling and allocation yield the DFG in Figure 3(b). Here, we can see that three functional units (FUs) are used. Operations *1 and *2 share FU1, operations *3, *4, and *5 share FU2, and operation +1, +2, and +3 share FU3. The sample period is 108 ns, each multiplication takes 20 ns, and each addition takes 10 ns. When $csteps$ is reduced from 10 to 9, instead of binding and scheduling from scratch, the algorithm reschedules based on current binding. For this example, after rescheduling, there are still timing violations for operations *3, *4, and *5 because they were all bound to FU2, as shown in Figure 3(c). Therefore, the split move is necessary in order to allow all operations to meet their timing requirements. A description of the split move follows.

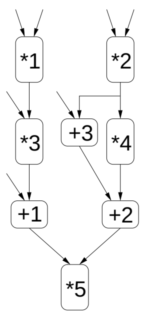
Based on the result of slack computation, we produce a graph including all the operations with negative slack. Each operation is represented by a node. In addition, there are three kinds of edges, defined as follows:

1) Data dependency edges, indicating that the destination node takes the source node's data as input;
2) Merge edges, indicating that the two nodes are currently bound to the same functional unit or same storage unit; and
3) Pseudo edges, used to restructure the graph for application of the min-cut algorithm. A pseudo source node and pseudo sink node are introduced into the graph. All input nodes are connected to the pseudo source node and all output nodes are connected to the pseudo sink node.

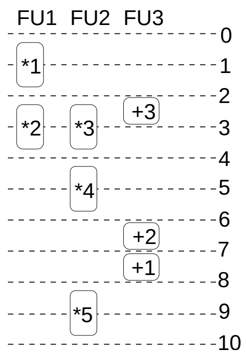
After constructing this graph, a min-cut algorithm is executed. First, an infinite capacity is assigned to all the pseudo edges and data dependency edges. Merge edges are each given capacities of one. If two nodes are connected by both a data dependency edge and a merge edge, the merge edge is eliminated because split moves on nodes sharing dependency edges do not improve the timing properties. Using a min-cut algorithm in this manner splits a minimal cardinality subset of nodes, allowing a reduction in the finish time of the ASAP. One could consider the impact of area and power consumption within the min-cut max-flow algorithm by weighting the edges appropriately. However, this would generally lead to to additional split operations, increasing CPU time.

Although decrementing $csteps$ may increase delay by at most one clock cycle, there may be some value of $csteps$ for which even fully parallel bindings do not allow an ASAP schedule to meet its timing constraints. Therefore, min-cut and rescheduling may not be carried out for some values of $csteps$. After the split move, the operations are rescheduled and slack is recomputed to determine whether timing constraints are met.

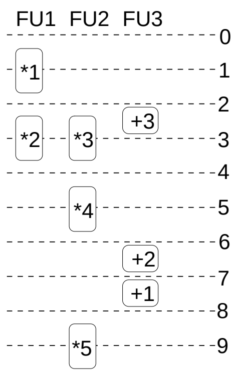
The algorithm described above was used to construct the graph shown in Figure 3(d). The dashed lines represent merge edges. The solid lines represent pseudo-edges and data dependence edges. Nodes S and T represent pseudo source and sink nodes respectively. After slack computation, we eliminate all the nodes and edges which are not on the critical path and assign a capacity of one to merge edges and a capacity of infinity to other edges, as shown in Figure 3(e). For this example, it is possible to cut through either edge (+2, +1), or edges (*3, *1) and (*4, *2). Here, we cut through +2 and +1, which is the minimal cut, thereby assigning +1 to a new functional unit, FU4. +3 and +2 remain bound to the original functional unit, FU3. As shown in Figure 3(f), no operation violates timing constraints.



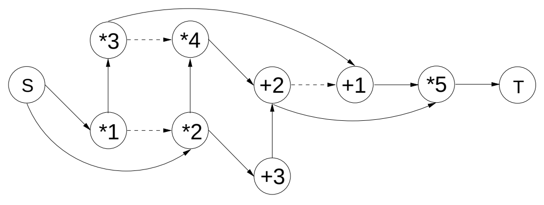
(a) CDFG



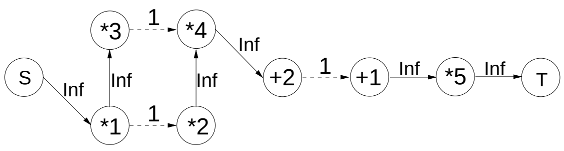
(b) Scheduling and allocation.



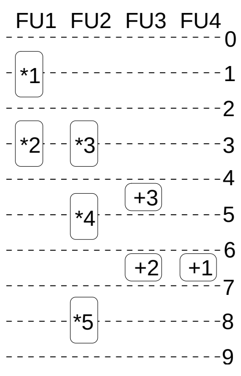
(c) Timing violation.



(d) Slack computation.



(e) Min-cut flow.



(f) After split move.

Fig. 3. Incremental changes on HLS.

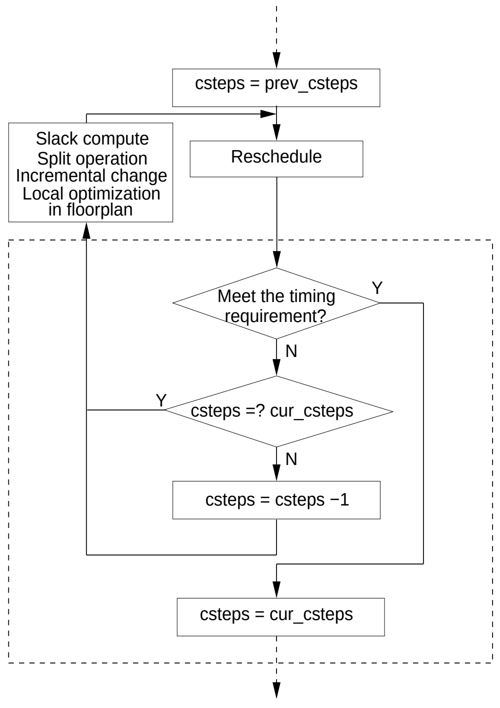


Fig. 4. Iterative split move for slack smaller than -1.

Another case must also be considered. If no valid solutions exist for the current value of $csteps$, IFP-HLS will skip further optimization and decrement $csteps$. IFP-HLS may reach a valid value of $csteps$ after repeatedly decrementing $csteps$. In this case, the slack values for some operations may less than -1. Hence, the value of $csteps$ is decremented and the split move, followed by rescheduling, are repeated until a valid solution is produced. This process is as shown in Figure 4.

IV. Incremental Floorplanning

As discussed in previous sections, in order to introduce incremental combined behavioral and physical optimization into high-level synthesis, a high-quality incremental floorplanner is necessary. We have tested this idea by building an incremental simulated annealing floorplanner into the IFP-HLS algorithm. In this section, we describe this incremental floorplanner.

This floorplanner handles blocks with different aspect ratios and produces non-slicing floorplans. Unlike the netlist partitioning approach used in ISCALP, it was designed primarily for quality, not speed. Although the impact on synthesis time would prevent incorporation of a conventional high-quality floorplanner in the inner loop of a high-level synthesis system, using incremental floorplanning enables both high quality and low synthesis time. High-level synthesis moves typically remove a single module or split a module into two. Therefore, many changes are small and their effects on the floorplan are mostly local. We reuse the modified previous floorplan as a starting point for each new floorplan. The previous floorplan is optimized. Therefore, re-optimization of the current floorplan to incorporate local changes is fast. In practice, we have found that this technique leads to quality-of-results and performance improvements over constructive floorplanning, even when compared with a very fast constructive floorplanner.

A. Floorplan Representation

The Adjacent Constraint Graph (ACG) floorplan representation is used within IFP-HLS's incremental floorplanner [36], [39], [40]. This representation will be briefly summarized here.

An ACG is a constraint graph satisfying the following three conditions: first, there is at most one relation (either horizontal or vertical) between any pair of vertices; second, there are no transitive edges; third, there are no *crosses*. A cross is a special edge configuration that can result in quadratic number of edges in the constraint graph. Figure 5 shows two cases of crosses and Figure 6 shows that a constraint graph with

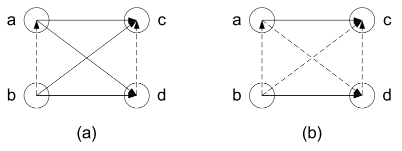


Fig. 5. (a) Horizontal cross and (b) vertical cross.

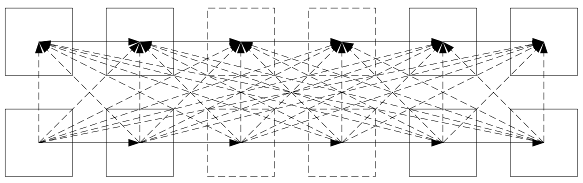


Fig. 6. A constraint graph without over-specifications and transitive edges can have quadratic number of edges.

crosses may have quadratic number of edges even when the first two conditions are met. It is proved that the number of edges in an ACG is at most $O\left(n^{1.5}\right)$ where n is the number of vertices [40].

The operations of removing and inserting vertices in an existing ACG are designed to reflect high-level binding decisions, i.e., merging and splitting. To obtain the physical position of each module, packing based on longest path computation is employed since the ACG itself is a constraint graph.

Perturbations on the graph are designed so that the ACG can be used in an iterative optimization heuristic such as simulated annealing (SA). They change the graph topology locally and have straightforward meanings in physical space. Since the interconnect lengths are determined by the physical positions of modules, which in turn depend on the graph topology, applying these perturbations changes the interconnects locally. Other perturbations include rotating a module and exchanging the two modules represented by any pair of vertices. The latter changes the interconnect globally.

B. Incremental Floorplanner

There are four situations in which the incremental floorplanner is called by the IFP-HLS framework. First, a floorplan should be generated after each ASAP schedule is produced. We call this an *initial* floorplanning. Second, a floorplan should be modified and optimized after each high-level synthesis move. We call this *per-move* floorplanning. Third, for each $csteps$ value, a floorplan for the best binding should be generated and compared to the existing best floorplans. We call this *per-cstep* floorplanning. Fourth, after determining the best clock frequency and binding, floorplanning is carried out to provide the final result. We call this *final* floorplanning.

Although initial, per-cstep, and final floorplanning are done with simulated annealing for quality, per-move floorplanning requires fewer global changes and less hill climbing. Moreover, perturbations resulting from high temperatures may disrupt high-quality floorplan structures. Therefore, it is reasonable to use lower temperatures for per-move floorplanning. In practice, we have found that using a temperature of zero results in good quality and performance. In other words, although simulated annealing is necessary in many cases, per-move floorplanning is done with a greedy iterative improvement algorithm.

The details of our approach follow. First, after generating the first ASAP schedule and binding, we have an initial set of modules and interconnections. Simulated annealing is used to obtain an initial floorplan. Since every interconnect net has exactly one driving module, multi-pin nets are broken into two-pin wires with the driving module as the source. The wire length is calculated as the Manhattan distance between the two modules connected by the wire. At this point, the unit-length switched capacitances of data transfers between two modules are available. We use these as weights for the wire lengths. The weighted total wire length is related to power consumption, i.e., optimizing weighted wire length minimizes interconnect power consumption. A weighted sum of the area and the interconnect power consumption is calculated as the floorplanner's cost function, that is,

$$A + w \sum_{e \in E} C_e D_e \tag{5}$$

where A is the area, w is the power consumption weight, E is the set of all wires, e is an interconnect wire, C_e is the unit-length switched capacitance for the data transfer along e, and D_e is the length of e. With this approach, we optimize the floorplan for both the interconnect power consumption and the area. The resulting floorplan will be improved during the consecutive incremental floorplanning high-level synthesis moves. Therefore, the number of simulated annealing iterations is bounded to reduce synthesis time.

After each high-level synthesis move, per-move floorplanning first modifies the previous floorplan by removing or splitting a module. The modules and switched capacitances are updated based upon the impact of these merges and splits. The floorplan is then re-optimized with a greedy iterative improvement algorithm using the same cost function as the simulated annealing algorithm. The greedy improvements are divided into consecutive rounds. In every round we apply the same number of perturbations to the floorplan. If less than 10% of the perturbations result in reduced costs, re-optimization stops. Although it would be easy to use a low simulated annealing temperature to allow some hill climbing during re-optimization, this was not necessary in practice. It should be pointed out here that changes to switched capacitances may require a few global changes in the ACG to obtain power consumption optimized floorplans. Therefore, we still allow the exchange perturbation to change the floorplan globally, but reduce its frequency to favor local perturbations.

When we find the best binding for a given value of $csteps$, we do per-cstep floorplanning and compare the result with the best floorplan from previous value of $csteps$. This time non-zero temperature simulated annealing is used because it increases floorplan quality. These normal simulated annealing runs occur only once per $csteps$ value, allowing their CPU costs to be amortized.

After determining the best binding across all the possible values of $csteps$, a final floorplanning run is carried out for that binding. This final floorplanning occurs only once per synthesis run. Therefore, it is acceptable to use a higher-quality, but slower, annealing schedule than those in the inner loop of high-level synthesis, thereby reducing chip area and interconnect power consumption.

During the annealing schedule, we use a constant multiplicative cooling factor, r, i.e.,

$$T' = r \times T \tag{6}$$

where T is the current temperature and T' is the temperature for the next iteration. The cooling factors for initial, per-cstep, and final floorplanning are 0.7, 0.8, and 0.9 respectively. At one temperature, if less than 10% of the perturbations are accepted, the annealing process stops. The ratio between the numbers of the perturbations at one temperature for initial, per-cstep, and final floorplanning is $1 : 1 : 5$. The number of perturbations per round for per-move floorplanning is the same as that in the final floorplanning.

The interconnect power consumption weight, w, is computed during synthesis for each floorplanning to avoid the difficulty of determining a proper value for all the situations. Before each floorplanning, we calculate the area-to-power-consumption ratio, w_0, using the existing floorplan, which is either the previous floorplan for per-move, per-cstep, and final floorplanning or the starting floorplan for initial floorplanning. For initial, per-cstep and final floorplanning, the weight w is set to $0.5 \cdot w_0$ to balance the area and the interconnect power consumption. For per-move floorplanning, it is more important to provide a prediction of the trend of interconnect power consumption in a limited time so that w is set to $2.5 \cdot w_0$ instead. Note that in this stage, not area cost but the prediction of the interconnect power consumption is the major consideration. Therefore, the wire length weight was set to be a big value compared to the area weight.

V. Experimental Results

In this section, we present the results produced by the IFP-HLS incremental floorplanning high-level synthesis algorithm described in Sections III and IV when run on a number of benchmarks. The results generated by ISCALP and IFP-HLS are compared. As explained in Section III-A, both approaches optimize area and power consumption. The experiments were conducted on Linux workstations with dual 933 MHz Pentium III processors and 512 MB of random access memory.

A. Benchmarks

We evaluated seventeen high-level synthesis benchmarks using a 0.18 µm technology library. *Chemical* and *IIR77* are infinite impulse response (IIR) filters used in industry. *DCT_IJPEG* is the Independent JPEG Group's implementation of digital cosine transform (DCT) [41]. *DCT_Wang* is a DCT algorithm named after the inventor [42]. Both DCT algorithms work on 8×8 arrays of pixels. *Elliptic*, an elliptic wave filter, comes from the NCSU CBL high-level synthesis benchmark suite [43]. *Jacobi* is the Jacobi iterative algorithm for solving a fourth order linear system [44]. *WDF* is a finite impulse response (FIR) wave digital filter. The largest benchmark is Jacobi with 24 multiplications, 8 divisions, 8 additions, and 16 subtractions. In addition, we generate five CDFGs using a pseudo random graph generator [45]. Random100 has 20 additions, 15 subtractions, and 19 multiplications. Random200 has 39 additions, 44 subtractions, and 36 multiplications. Random300 has 59 additions, 58 subtractions, and 72 multiplications.

IFP-HLS had better performance than ISCALP on these large randomized benchmarks. In order to determine whether the improved performance of IFP-HLS was the result of random graph structure or benchmark size, we generated two structured benchmarks, Small and Serial. Small is composed of five operations connected in parallel. Serial is composed of 45 operations connected in serial. As shown in Table I, IFP-HLS has better CPU time for structured the large benchmark Serial. This is consistent with the results for other other large benchmarks.

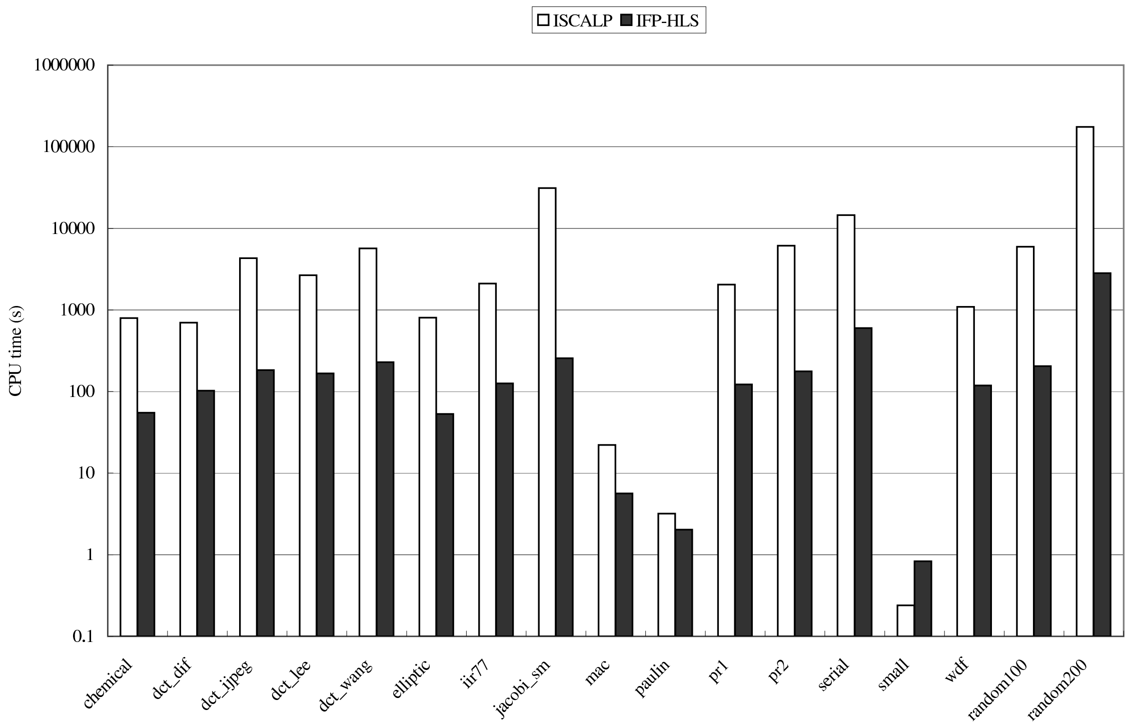
The area of each benchmark described in this section was estimated using pre-synthesized functional-units (e.g., adders, multipliers, etc.) based on NEC's 0.18 µm process and the floorplanner from high-level synthesis tool. The logic power consumption of each benchmark was evaluated using power models from the pre-synthesized functional-unit level design library. A full-system switching activity simulator was used during power consumption computation. Wire power consumption and wire delay were calculated based on the wire capacitances estimated using Cong's and Pan's technique [46] and the wire length information from floorplanner of high-level synthesis design tools. As described in Section III, both logic and wire delays were calculated to determine whether each design meets its timing requirements. However, since the wire delay estimation is only implemented in IFP-HLS; this function was not used when comparing to ISCALP.

B. Results

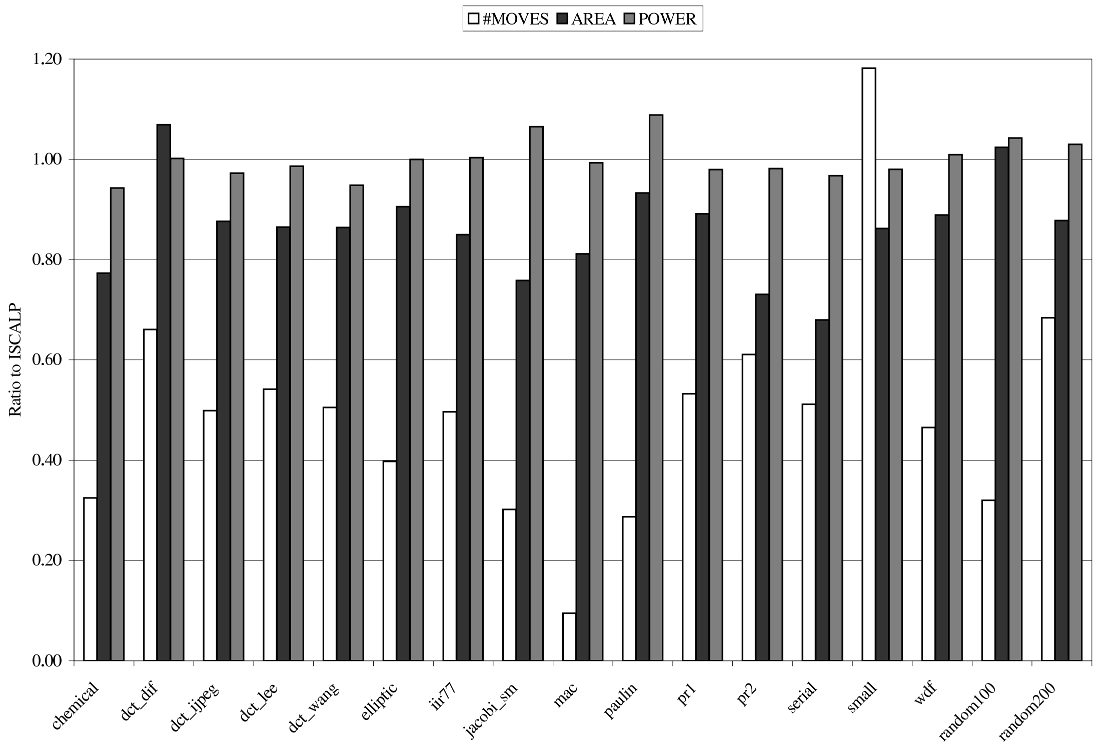
The results of running ISCALP and IFP-HLS on non-unity aspect ratio functional units are shown in Figure 7. As shown in the Figure 7(a), Figure 7(b), and Table I, IFP-HLS achieves an average CPU time speedup of 24.72×, 13.76% improvement in area, and 50% reduction in the number of merge move in comparison with ISCALP. Low power consumption is maintained.

ISCALP uses a constructive floorplanner that may suffer performance degradation when used with non-unity aspect ratio functional units. In order to determine whether the improvement in quality and run time were the result of the specific type of floorplanner used in ISCALP, we repeated all experiments using only unity aspect ratio functional units. As shown in Figure 8, Table I, and Table III, the IFP-HLS algorithm achieves an average CPU time speedup of 2.03×, 11.32% improvement in area, and 54% reduction in the number of merge move, while maintaining the same low power consumption as ISCALP.

As shown in Figure 7, Figure 8, Table I, and Table III, IFP-HLS always has better CPU time than ISCALP for both non-unity and unity aspect ratio cases except for two very small unity aspect ratio benchmarks (PAULIN and MAC).



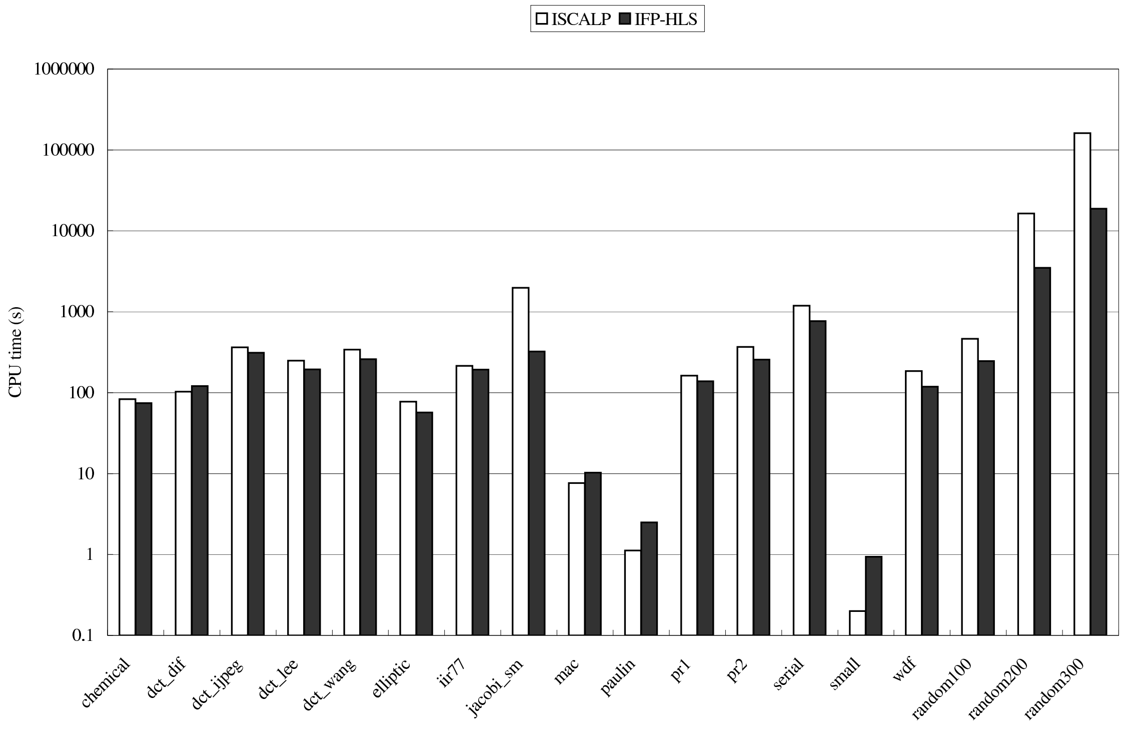
(a) CPU time for non-unity aspect ratio functional units.



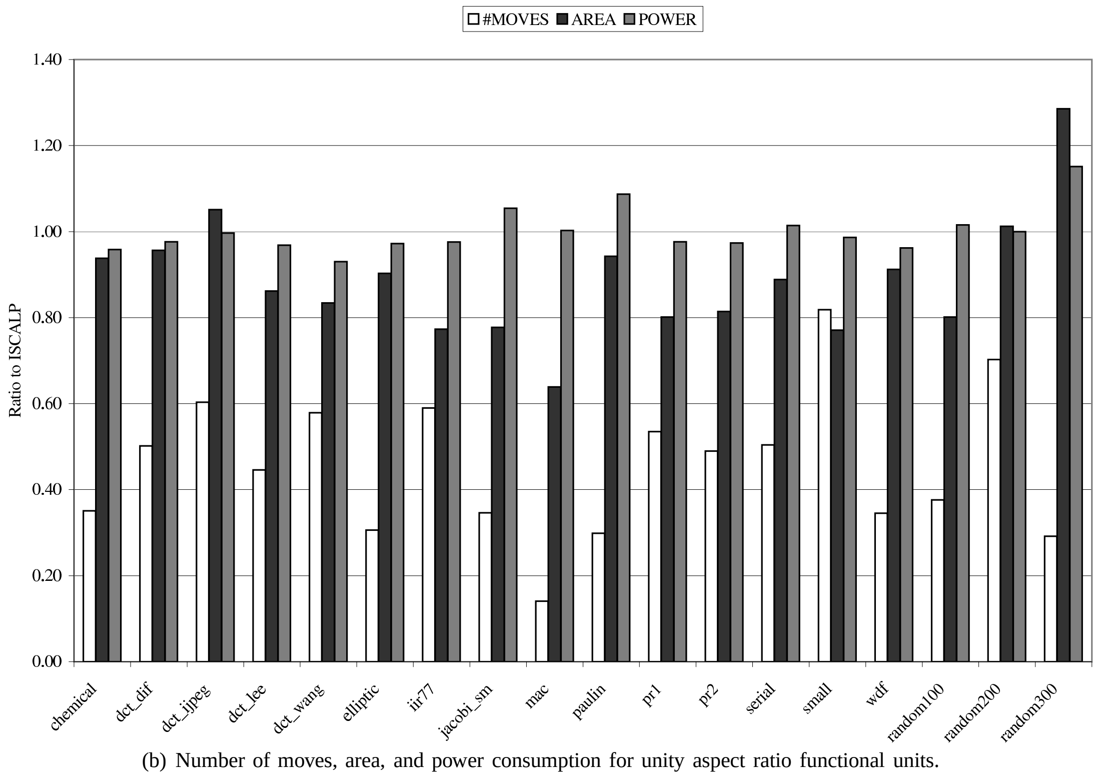
(b) Number of moves, area, and power consumption for non-unity aspect ratio functional units.

Fig. 7. Comparison between ISCALP & IFP-HLS for non-unity aspect ratio functional units.

There are two contributors to CPU time (as shown in Equation 3): the number of high-level synthesis moves and the resulting floorplanning operations. ISCALP employs a fast constructive slicing floorplanner based on netlist partitioning and rotation/orientation selection to obtain a floorplan optimized for wire length and area. It is faster than our simulated annealing floorplanner for small benchmarks with only a few blocks largely due to its determinism. The sim-



(a) CPU time for unity aspect ratio functional units.



(b) Number of moves, area, and power consumption for unity aspect ratio functional units.

Fig. 8. Comparison between ISCALP & IFP-HLS for unity aspect ratio functional units.

TABLE I

NUMBERS OF MERGES AND CPU TIMES OF DIFFERENT BENCHMARKS

Benchmark	Unity aspect ratio						Non-unity aspect ratio					
	No. of Merges		No. of Splits	CPU Time			No. of Merges		No. of Splits	CPU Time		
	ISCALP	IFP-HLS	IFP-HLS	ISCALP (s)	IFP-HLS (s)	Speedup (×)	ISCALP	IFP-HLS	IFP-HLS	ISCALP (s)	IFP-HLS (s)	Speedup (×)
CHEMICAL	593	208	24	83.35	74.19	1.12	585	190	26	793.83	55.15	14.39
DCT_DIF	981	492	1	102.98	120.86	0.85	769	508	1	699.96	102.27	6.84
DCT_IJPEG	630	380	0	363.12	310.91	1.17	850	424	11	4297.13	183.36	23.44
DCT_LEE	1512	674	2	248.07	194.41	1.28	1276	691	5	2669.18	166.56	16.03
DCT_WANG	974	564	10	340.40	259.53	1.31	1019	515	3	5678.98	229.11	24.79
ELLIPTIC	562	172	16	77.46	57.04	1.36	533	212	11	804.29	53.16	15.13
IIR77	858	506	2	214.89	192.87	1.11	858	426	1	2102.27	126.04	16.68
JACOBI_SM	1652	572	52	1982.55	322.97	6.14	1755	530	26	31187.15	256.60	121.54
MAC	220	31	13	7.64	10.28	0.74	200	19	14	22.12	5.64	3.92
PAULIN	87	26	6	1.12	2.50	0.45	87	25	6	3.20	2.03	1.58
PR1	839	449	10	162.79	138.13	1.18	841	448	12	2041.15	121.94	16.74
PR2	1074	526	20	366.87	256.12	1.43	866	529	27	6145.86	177.36	34.65
SERIAL	5200	2620	3	1187.37	767.48	1.55	5200	2660	1	14503.25	599.59	24.19
SMALL	11	9	0	0.20	0.94	0.21	11	13	2	0.24	0.83	0.29
WDF	1827	631	6	185.52	118.74	1.56	1588	739	9	1092.92	118.92	9.19
RANDOM100	1359	511	10	462.14	246.24	1.88	1353	433	5	5951.79	204.52	29.10
RANDOM200	1110	780	1	16438.33	3498.53	4.70	1140	780	2	174540.95	2826.22	61.76
RANDOM300	2810	820	0	160997.92	18786.70	8.57	N/A*	900	2	N/A*	12650.64	N/A*
Average	1238.83	553.94	9.78	10179.04	1408.80	2.03	1113.59*	537.76*	9.11	14854.96*	307.61*	24.72*

*To solve non-unity aspect ratio Random300, ISCALP had not yet halted after 120 hours. The non-unity aspect ratio Random300 benchmark was excluded from the computation for average numbers.

TABLE II

CPU TIMES BREAK DOWN OF DIFFERENT BENCHMARKS

Benchmark	Unity aspect ratio						Non-unity aspect ratio					
	ISCALP			IFP-HLS			ISCALP			IFP-HLS		
	T_{fp} (s)	T_{total} (s)	Ratio* (%)	T_{fp} (s)	T_{total} (s)	Ratio* (%)	T_{fp} (s)	T_{total} (s)	Ratio* (%)	T_{fp} (s)	T_{total} (s)	Ratio* (%)
CHEMICAL	61.19	83.35	73.41	61.24	74.19	82.54	770.91	793.83	97.11	43.60	55.15	79.06
DCT_DIF	75.57	102.98	73.38	101.86	120.86	84.28	674.93	699.96	96.42	82.95	102.27	81.11
DCT_IJPEG	308.59	363.12	84.98	269.28	310.91	86.61	4238.01	4297.13	98.62	146.66	183.36	79.98
DCT_LEE	191.11	248.07	77.04	159.66	194.41	82.13	2616.96	2669.18	98.04	130.88	166.56	78.58
DCT_WANG	275.61	340.40	80.97	214.17	259.53	82.52	5613.21	5678.98	98.84	185.94	229.11	81.16
ELLIPTIC	49.42	77.46	63.80	47.08	57.04	82.54	780.94	804.29	97.10	41.44	53.16	77.95
IIR77	168.91	214.89	78.60	164.32	192.87	85.20	2056.25	2102.27	97.81	104.88	126.04	83.21
JACOBI_SM	1846.04	1982.55	93.11	245.88	322.97	76.13	31029.15	31187.15	99.49	188.11	256.60	73.31
MAC	4.80	7.64	62.83	9.28	10.28	90.27	19.42	22.12	87.79	4.89	5.64	86.70
PAULIN	0.54	1.12	48.21	2.11	2.50	84.40	2.50	3.20	78.13	1.66	2.03	81.77
PR1	126.84	162.79	77.92	112.32	138.13	81.31	2003.70	2041.15	98.17	96.38	121.94	79.04
PR2	299.53	366.87	81.64	213.68	256.12	83.43	6085.16	6145.86	99.01	141.40	177.36	79.72
SERIAL	974.21	1187.37	82.05	634.26	767.48	82.64	14288.12	14503.25	98.52	476.38	599.59	79.45
SMALL	0.09	0.20	45.00	0.84	0.94	89.36	0.15	0.24	62.50	0.71	0.83	85.54
WDF	126.58	185.52	68.23	90.21	118.74	75.97	1037.36	1092.92	94.92	88.49	118.92	74.41
RANDOM100	383.64	462.14	83.01	208.30	246.24	84.59	5869.24	5951.79	98.61	172.87	204.52	84.52
RANDOM200	15484.14	16438.33	94.20	2877.49	3498.53	82.25	173591.19	174540.95	99.46	2220.85	2826.22	78.58
RANDOM300	151251.05	160997.92	93.95	15798.36	18786.70	84.09	N/A**	N/A**	N/A**	10089.21	12650.64	79.75
Average	9534.88	10179.04	75.69	1178.35	1408.80	83.35	14745.72**	14854.96**	94.15**	242.83**	307.61**	80.24**

* Ratio = T_{fp}/T_{total}

**To solve non-unity aspect ratio Random300, ISCALP had not yet halted after 120 hours. The non-unity aspect ratio Random300 benchmark was excluded from the computation for average numbers.

ulated annealing algorithm may re-visit same valid solutions multiple times before reaching the halting conditions while constructive slicing floorplanner can quickly consider all slicing structure floorplanners, given small enough problem sizes. In contrast, the simulated annealing floorplanner is relatively faster on large problem instances because it can focus its moves on the most promising regions of the solution space while the constructive floorplanner is left to explicitly consider an exponentially-increasing number of points in the solution space. Please note that both floorplanners run quickly on small benchmarks. We are primarily concerned with floorplanner performance on large problem instances, for which run-time is a concern. In addition, recall that ISCALP is an interconnect-aware, power-driven high-level synthesis tool. These results show that, on average, IFP-HLS achieves better CPU time and area while maintaining good power consumption. We also analysis the time break down between high-level synthesis moves and floorplanning. As shown in Table II, floorplanning used more than 75.69% of the total CPU time on average for both ISCALP and IFP-HLS; floorplanning is the most time-consuming part of the high-level synthesis design flow.

In an attempt to isolate the impact of using a constructive

TABLE III
AREA AND POWER IMPROVEMENTS OF DIFFERENT BENCHMARKS

Benchmark	Area Improvement (%)		Wire Power Improvement (%)		Total Power Improvement (%)	
	Unity	Non-unity	Unity	Non-unity	Unity	Non-unity
CHEMICAL	6.23	22.67	22.69	9.39	4.17	5.72
DCT_DIF	4.36	-6.91	41.51	21.86	2.37	-0.16
DCT_IJPEG	-5.09	12.35	49.59	-4.39	0.35	2.76
DCT_LEE	13.84	13.53	22.74	7.58	3.16	1.35
DCT_WANG	16.60	13.61	38.50	42.82	7.00	5.15
ELLIPTIC	9.70	9.45	22.18	28.32	2.77	0.03
IIR77	22.69	15.03	21.32	-0.37	2.42	-0.34
JACOBI_SM	22.26	24.17	14.13	9.76	-5.41	-6.51
MAC	36.13	18.84	9.65	13.72	-0.27	0.70
PAULIN	5.74	6.72	26.19	23.39	-8.72	-8.86
PR1	19.86	10.87	6.13	16.44	2.39	2.05
PR2	18.56	26.93	30.83	-7.75	2.67	1.84
SERIAL	11.16	32.04	11.74	31.57	-1.42	3.28
SMALL	22.89	13.78	36.93	48.83	1.36	1.98
WDF	8.79	11.10	9.87	-12.50	3.79	-0.91
RANDOM100	19.88	-2.41	14.67	19.94	-1.55	-4.24
RANDOM200	-1.22	12.20	19.45	-1.90	0.01	-3.01
RANDOM300	-28.55	N/A*	31.80	N/A*	-15.11	N/A*
Average	11.32	13.76*	23.89	14.51*	0.00	0.05*

*To solve non-unity aspect ratio Random300, ISCALP had not yet halted after 120 hours. The non-unity aspect ratio Random300 benchmark was excluded from the computation for average numbers.

floorplanner from the impact of using incremental optimization, we compared the results produced by running ISCALP followed by a high-quality simulated annealing floorplanner by those produced by IFP-HLS. On average, this results in a 1.6% increase in area and 2.7% decrease in total power compared to IFP-HLS for unity aspect ratio functional units and a 0.8% increase in area and 1.3% decrease in total power consumption for non-unity aspect ratio functional units. Note that ISCALP aggressively optimizes power consumption. These results indicate that the incremental optimization algorithm within IFP-HLS permits comparable quality, using much less CPU time, compared to a non-incremental behavioral synthesis algorithm followed by an iterative improvement floorplanner.

VI. CONCLUSIONS

This article presented an incremental floorplanning, high-level synthesis system that integrates high-level and physical-level design algorithms to concurrently improve a design's schedule, resource binding, and floorplan. Compared with previous approaches that repeatedly call loosely coupled floorplanners, this approach has the benefit of efficiency, stability, and better quality results. As shown in Section V, for non-unity aspect functional units, incremental floorplanning allowed an average CPU time speedup of $24.72\times$ and an area improvement of 13.76%. For unity aspect ratio functional units, the CPU time speedup was $2.03\times$ and area was improved by 11.32%. In both cases, the low power consumption of a state-of-the-art, low-power, interconnect-aware high-level synthesis algorithm was maintained. We conclude that incremental floorplanning improved the quality of synthesis results and improves performance dramatically, making synthesis from large specifications practical.

VII. ACKNOWLEDGMENTS

We would like to thank Prof. Niraj Jha at Princeton University for access to ISCALP and NEC Labs America for access to their 0.18 µm technology library. We would also like to thank Yongpan Liu at Tsinghua University, Dr. Anand Raghunathan at NEC Labs America, and Prof. Lin Zhong at Rice University for their helpful suggestions.

REFERENCES

[1] R. Camposano and W. Wolf, *High Level VLSI Synthesis*. Kluwer Academic Publishers, MA, 1991.

[2] D. C. Ku and G. D. Micheli, *High Level Synthesis of ASICs Under Timing and Synchronization Constraints*. Kluwer Academic Publishers, MA, 1992.

[3] D. Gajski, et al., *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, MA, 1992.

[4] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, MA, 1997.

[5] P. G. Paulin, J. P. Knight, and E. F. Girczyc, "HAL: A multi-paradigm approach to automatic data path synthesis," in *Proc. Design Automation Conf.*, June 1986, pp. 263–270.

[6] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computers*, vol. 10, no. 3, pp. 29–41, Sept. 1993.

[7] R. Mehra and J. Rabaey, "Behavioral level power estimation and exploration," in *Proc. Int. Wkshp. on Low Power Design*, Apr. 1994, pp. 197–202.

[8] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," in *Proc. Int. Symp. Low-Power Design*, Apr. 1994.

[9] L. Goodby, A. Orailoglu, and P. M. Chau, "Microarchitecture synthesis of performance-constrained, low-power VLSI designs," in *Proc. Int. Conf. Computer Design*, Oct. 1994.

[10] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," in *Proc. Int. Conf. Computer Design*, Oct. 1994, pp. 318–322.

[11] A. P. Chandrakasan, et al., "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 12–31, Jan. 1995.

[12] R. S. Martin and J. P. Knight, "Power profiler: Optimizing ASICs power consumption at the behavioral level," in *Proc. Design Automation Conf.*, June 1995.

[13] J. M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proc. Design Automation Conf.*, June 1995.

[14] N. Kumar, et al., "Profile-driven behavioral synthesis for low-power vlsi systems," *IEEE Des. Test*, vol. 12, no. 3, pp. 70–84, 1995.

[15] A. Raghunathan and N. K. Jha, "SCALP: An iterative-improvement-based low-power data path synthesis system," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1260–1277, Nov. 1997.

[16] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, “High-level synthesis of low power control-flow intensive circuits,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1715–1729, Dec. 1999.

[17] H. P. Peixoto and M. F. Jacome, “A new technique for estimating lower bounds on latency for high level synthesis,” in *Proc. Great Lakes Symp. VLSI*, Mar. 2000, pp. 129–132.

[18] M. C. McFarland and T. J. Kowalski, “Incorporating bottom-up design into hardware synthesis,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 9, pp. 938–950, Sept. 1990.

[19] D. W. Knapp, “Fasolt: A program for feedback-driven data-path optimization,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 6, pp. 677–695, June 1992.

[20] J. P. Weng and A. C. Parker, “3D scheduling: High-level synthesis with floorplanning,” in *Proc. Design Automation Conf.*, June 1992.

[21] Y. M. Fang and D. F. Wong, “Simultaneous functional-unit binding and floorplanning,” in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994.

[22] M. Xu and F. J. Kurdahi, “Layout-driven rtl binding techniques for high-level synthesis using accurate estimators,” *ACM Trans. Design Automation Electronic Systems*, vol. 2, no. 4, pp. 312–343, Oct. 1997.

[23] W. E. Dougherty and D. E. Thomas, “Unifying behavioral synthesis and physical design,” in *Proc. Design Automation Conf.*, June 2000.

[24] P. G. Paulin and J. P. Knight, “Scheduling and binding algorithms for high-level synthesis,” in *Proc. Design Automation Conf.*, June 1989, pp. 1–6.

[25] C. A. Papachristou and H. Konuk, “A linear program driven scheduling and allocation method followed by an interconnect optimization algorithm,” in *Proc. Design Automation Conf.*, June 1990.

[26] T. A. Ly, W. L. Elwood, and E. F. Girczyc, “A generalized interconnect model for data path synthesis,” in *Proc. Design Automation Conf.*, June 1990.

[27] S. Tarafdar and M. Leeser, “The DT-model: High-level synthesis using data transfer,” in *Proc. Design Automation Conf.*, June 1998.

[28] C. Jego, E. Casseau, and E. Martin, “Interconnect cost control during high-level synthesis,” in *Proc. Int. Conf. Design Circuits Integrated System*, Nov. 2000.

[29] R. Ho, K. Mai, and M. Horowitz, “The future of wires,” *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, 2001.

[30] P. Prabhakaran and P. Banerjee, “Simultaneous scheduling, binding and floorplanning high-level synthesis,” in *Proc. Int. Conf. VLSI Design*, Jan. 1998.

[31] L. Zhong and N. K. Jha, “Interconnect-aware low power high-level synthesis,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 336–351, Mar. 2005.

[32] A. Stammermann, et al., “Binding, allocation and floorplanning in low power high-level synthesis,” in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2003.

[33] O. Coudert, et al., “Incremental CAD,” in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 236–244.

[34] W. Choi and K. Bazargan, “Hierarchical global floorplacement using simulated annealing and network flow migration,” in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 2003.

[35] Z. P. Gu, et al., “Incremental exploration of the combined physical and behavioral design space,” in *Proc. Design Automation Conf.*, June 2005, pp. 208–213.

[36] H. Zhou and J. Wang, “ACG–Adjacent constraint graph for general floorplans,” in *Proc. Int. Conf. Computer Design*, Oct. 2004.

[37] C. M. Fiduccia and R. M. Mattheyses, “A linear-time heuristic for improving network partitions,” in *Proc. Design Automation Conf.*, June 1982, pp. 173–181.

[38] L. Stockmeyer, “Optimal orientations of cells in slicing floorplan designs,” *Information & Control*, vol. 57, no. 2/3, pp. 91–101, May 1983.

[39] J. Wang and H. Zhou, “Interconnect estimation without packing via ACG floorplans,” in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 2005.

[40] J. Wang, “Floorplanning by adjacent constrain graph and its applications,” Master’s thesis, Northwestern University, June 2005.

[41] “Independent JPEG group,” www.ijp.org.

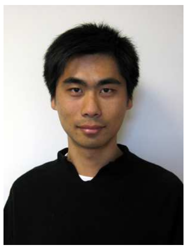
[42] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic, NY, 1990.

[43] “NCSU CBL high-level synthesis benchmark suite,” www.cbl.ncsu.edu/benchmarks.

[44] S. Y. Kung, *VLSI Array Processors*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[45] R. P. Dick, D. L. Rhodes, and W. Wolf, “TGFF: Task graphs for free,” in *Proc. Int. Wkshp. Hardware/Software Co-Design*, Mar. 1998, pp. 97–101.

[46] J. Cong and Z. Pan, “Interconnect performance estimation models for design planning,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 739–752, June 2001.



Zhenyu (Peter) Gu (S'04) received his B.S. and M.S. degrees from Fudan University, China in 2000 and 2003. He is currently a Ph.D. student at Northwestern University's Department of Electrical Engineering and Computer Science. Gu has published in the areas of behavioral synthesis and thermal analysis of integrated circuits.

Jia Wang received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, China, in 2002, and is currently working toward the Ph.D. in computer engineering at Northwestern University, Evanston, IL. His research interests are on VLSI computer-aided design, especially algorithm design.

Robert P. Dick (S'95-M'02) received his B.S. degree from Clarkson University and his Ph.D. degree from Princeton University. He worked as a Visiting Researcher at NEC Labs America, a Visiting Professor at Tsinghua University's Department of Electronic Engineering, and is currently an Assistant Professor at Northwestern University's Department of Electrical Engineering and Computer Science. Robert received an NSF CAREER award and won his department's Best Teacher of the Year award in 2004. He has published in the areas of embedded system synthesis, mobile ad-hoc network protocols, reliability, behavioral synthesis, data compression, embedded operating systems, and thermal analysis of integrated circuits.

Hai Zhou (M'04-SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Bejing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from University of Texas, Austin, in 1999.

Before he joined the faculty of Northwestern University, he was with the Advanced Technology Group, Synopsys, Inc., Mountain View, CA. He is currently an Assistant Professor of electrical engineering and computer science at Northwestern University, Evanston, IL. His research interests include very large scale integrated computer-aided design, algorithm design, and formal methods.

Dr. Zhou served on the technical program committees of many conferences on very large scale integrated circuits and computer-aided design. He is a recipient of the CAREER Award from the National Science Foundation in 2003.