PAPER Special Section on Reconfigurable Systems

Low-Power Loop Parallelization onto CGRA Utilizing Variable Dual V_{DD}*

Bing XU[†], ***Nonmember***, **Shouyi YIN**[†a)], ***Member***, **Leibo LIU**[†], ***and*** **Shaojun WEI**[†], ***Nonmembers***

SUMMARY Coarse Grained Reconfigurable Architectures (CGRAs) are promising platform based on its high-performance and low cost. Researchers have developed efficient compilers for mapping compute-intensive applications on CGRA using modulo scheduling. In order to generate loop kernel, every stage of kernel are forced to have the same execution time which is determined by the critical PE. Hence non-critical PEs can decrease the supply voltage according to its slack time. The variable Dual-V_{DD} CGRA incorporates this feature to reduce power consumption. Previous work mainly focuses on calculating a global optimal V_{DDL} using overall optimization method that does not fully exploit the flexibility of architecture. In this brief, we adopt variable optimal V_{DDL} in each stage of kernel concerning their pattern respectively instead of the fixed simulated global optimal V_{DDL}. Experiment shows our proposed heuristic approach could reduce the power by 27.6% on average without decreasing performance. The compilation time is also acceptable.

key words: *loop mapping, software pipelining, Dual-V_{DD}, low power, Graph Minor*

1. Introduction

Portable devices faced more and more challenges for simultaneous demands for high flexibility and high performance as well as low power consumption than the traditional processor. Field Programmable Gate Arrays (FPGAs) were a suitable choice in portable domain in past days because it had relative high flexibility and performance only with disadvantage of high power consumption for managing the fine-grain reconfigurable units. Coarse Grained Reconfigurable Architectures has been attracting the industry and academia for its low power consumption, high flexibility and performance obtained from coarse grain and programmability. CGRA are becoming more and more prevalent in the portable environment.

As Shown in Fig. 1, ADRES-like [2] CGRA typically consists of three main components: a 2-D mesh coarse-grained reconfigurable elements (PE) array (PEA), a data memory and context memory, a host controller which attaches the CGRA to host processor. The size and memory width of CGRA can varies quite different. A PE basically includes an ALU and a register file as depicted in the right part of Fig. 1. According to the distributed register feature and the mesh interconnect by which each PE is connected only to the neighbors, the CGRA compiler is much more complex than the traditional compiler because it should take the Placement and Routing (P&R) into account.

To accelerate the compute-intensive application, the common method was to parallelize loops because the loops are the most significant time consuming part. It can be innermost loop parallelization or nest loop parallelization [9]. Parallelization of innermost loop was explicit pipelining of fixed schedule which was referred to as software pipelining and modulo scheduling [1]. Initial Interval (*II*) was used as the performance metric to guide the loop transformation. *II* was defined as the number of cycles between the start of two consecutive iterations of a loop and was equal to the equivalent execution time of a single iteration if the kernel contained only one copy of iteration. In this view, we can easily understand the shorter *II* the higher performance.

Previous work had made significant contributions to reduce the *II*. For example, in [4], Park offered a Node-centric Scheduling by focusing on assigning DFG nodes to PEs, where the placement problem was the first priority and the routing was the by product. Then Park improved his method by selecting intelligent paths from source to sink PEs that would not block other operand paths. The improved one was called Edge-centric Modulo Scheduling (EMS) [6]. EMS gave priority to routing rather than placement.

Then a more efficient graph-based method called EPIMap [7] was introduced by Hamzeh. He described mapping problem as finding a subgraph in a minimally Time-Extended CGRA (TEC) graph, where the subgraph is Epimorphic to the input DFG. EPIMap was categorized into temporal mapping [3]. Operands in same stage were data independent and were not connected with each other. In EPIMap, *II* were defined as the stage numbers of explicit kernel in TEC graph. The shorter the *II*, the shorter the TEC graph hence the higher the performance.

Chen developed another graph-based method based on Graph Minor Testing Approach [11]. He described modulo scheduler as *II* different configurations, each of them corresponded to a particular cycle in the kernel. Instead of using TEC graph, he restricted the time axis to the target *II* to get a simplified graph as schedule and route graph (SRG) by adding wrap around edges from the last cycle to the first cycle. Then he extended Undirected Graph Minor to

Manuscript received May 5, 2014.
Manuscript revised September 5, 2014.
Manuscript publicized November 19, 2014.

[†]The authors are with Institute of Microelectronic, Tsinghua University, Beijing, China.

*This work is supported in part by the China Major S&T Project (No.2013ZX01033001-001-003), the International S&T Cooperation Project of China grant (No. 2012DFA11170), the Tsinghua Indigenous Research Project (No.20111080997), the NNSF of China grant (No.61274131) and the China 863 Program (No.2012AA012701).

a) E-mail: yinsy@tsinghua.edu.cn

DOI: 10.1587/transinf.2014RCP0004

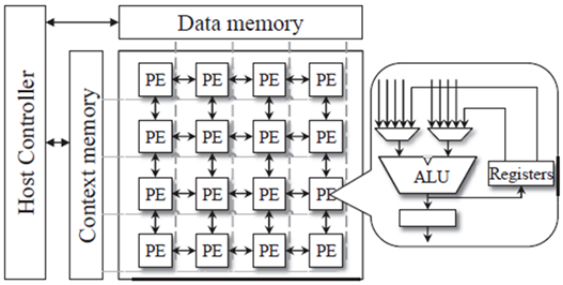

Fig. 1 The CGRA template.

directed case and used this math tool to mapping DFG onto SRG. Since this method focused only on the steady stage of pipelining where the CGRA consumed the most power, we could modify this method to reduce the power consumption without decreasing the performance.

Formulating explicit kernel requires that all stages of kernel have the same execution time which is determined by the critical PEs path (e.g. multiplier). There are slack time on non-critical PEs path (e.g. not, add) because these PEs are much faster. According to [5], we can execute these operations on lower voltage in a fixed Dual-V_{DD} CGRA, because the timing constraint is not violated by the additional delay due to level shifters and increased delay on lower voltage.

Zhu appealed to a Variable Dual-V_{DD} CGRA [8] in which he calculated unique optimal V_{DDL} for different applications. It was an evolution version of [5] in which the Dual-V_{DD} voltage were empirically setting to **1V** and **0.7V.** Estimation of V_{DDL} was categorized into two methods: relative slow simulation method and much more quick technique using path delay distribution [10]. We choose the former one because its compilation time is acceptable.

We applied the Dual-V_{DD} architecture incorporated with new algorithm into loop mapping domain. For a better mapping schedule which offered sharp power reduction without affecting the performance, this brief made three contributions:

Apply Dual-V_{DD} optimizing to loop parallelization: Previous research fails to reduce power in cases with high speed execution efficiency.

Unique optimal V_{DDL} for each stage of kernel instead of a fixed global optimal V_{DDL} for the whole kernel itself: In modulo scheduling method, different stage of kernel usually has different operands distribution thus can has different optimal V_{DDL}. We can achieve higher power efficiency by evolving from Zhu's global granularity to stage-level granularity.

Revised Graph Minor framework: Adding recomputation and routing techniques, which one to choose depends on power evaluation function obtained from above, of EPIMap to Graph Minor to reshape the original DFG for accelerating the mapping. Then we are able to choose the lower power consuming one between two schedulers with same *II* and different patterns. Since the core routine of Graph Minor does not change, we can reduce power consumption without affecting performance.

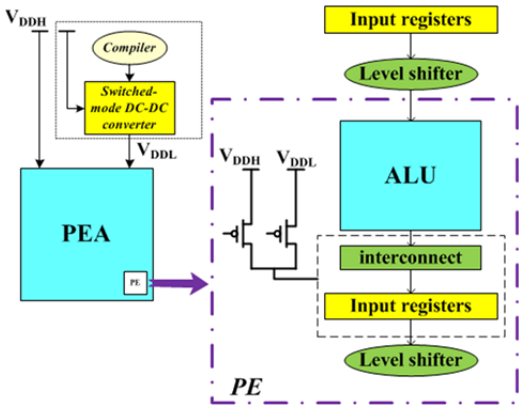

Fig. 2 Variable Dual-V_{DD} PEA.

The rest of paper is organized as follows. Section 2 describes the adopted variable Dual V_{DD} CGRA. Section 3 depicts rough process of our method. Section 4 gives the detailed math formulation and cost function. The experiment result is discussed in Sect. 5. At last, we conclude in Sect. 6.

2. Architecture

As shown in above, we should make full use of slack time of the non-critical PE. In this brief, we adopt the ADRES-like CGRA template which is modified with variable Dual V_{DD} function. The architecture has been raised in [8] which depicted in Fig. 2.

The sum of delay of ALU and delay of interconnect is the total execution time. We could make a configurable dual V_{DD} switch to the both components. However, the ALU is the critical part (75%) concerning delay while interconnect is the critical part (90%) concerning power, whose POWER/DELAY is tens of times larger than ALU through simulation results. Thus the configurable dual V_{DD} is designed for interconnect rather than ALU. Two selecting transistors and two-bit configurations are used to choose higher power supply V_{DDH} or lower power supply V_{DDL}. Level shifters are required at the bottom where the V_{DDL} signals are transmitted to the V_{DDH} domain.

An adjustable dc-dc converter is a necessary cost to generate the variable V_{DDL}. The switched converter is controlled by the compiler which can take advantage of our proposed mapping method while the fixed Dual V_{DD} architecture is inadaptable to achieve the best power reduction ratio of various applications. Zhu utilize this architecture to get unique global optimized V_{DDL} for given application as a whole, while we delve further and calculate unique optimized V_{DDL} for each stage of kernel when applying our modulo scheduling parallelization.

3. Motivation and Main Idea

We give examples to show main idea of our last two contributions. Firstly, power consumption of stage-level control

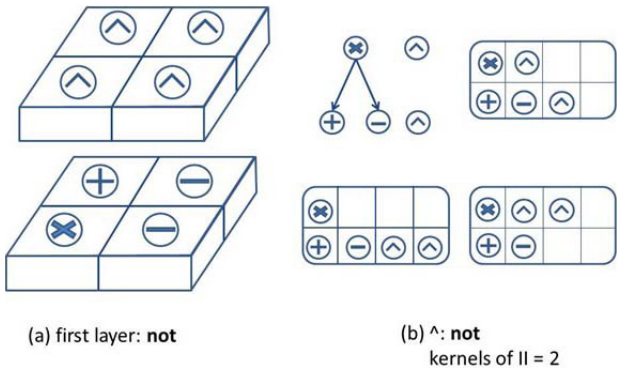

Fig. 3 Mapping patterns can affect power consumption.

is possibly lower than that of global granularity in Fig. 3 a. According to [5], we can execute faster operations on lower voltage. In order to explain the essential idea, we make an extremely idealized case based on our simulation results. The execution time of *multiplier: shifter: adder: not* is about *12: 6: 4: 1*. The supply voltage of critical PE (multiplier) is setting to the V_{DDH} while other PEs (shifter, adder, not) can adjust to different supply voltage accordingly. A lower voltage means a longer execution time. We set the V_{DDH} as *1V* and accept *1V:0.9V:0.7V:0.5V* as minimum permissible supply voltage V_{DDL} of each operand for convenient. For example, if we apply *0.6V* to *adder*, the execution time of adder is definitely longer than 12, which violate the time constraints.

Power is proportional to square of voltage. We use the sum of all the PE's square of voltage to represent the power for illustrating the idea. In Fig. 3 a, four PEs of first layer are all assigned to *not* operands, the second layer are *multiplier* and *adder*. Using two granularity control of voltage, global-level and stage-level, we calculate power consumption reduction ratio respectively.

We first apply the global granularity voltage control. In this case, there has only one fixed V_{DDL} for all PEs. If we set *0.7V* as the V_{DDL}, then all PEs except the multiplier can use the V_{DDL}. If we set *0.5V* as the V_{DDL}, only the first layer can use the lower voltage because the adder would violate time constraints. Then we appeal to stage-level granularity voltage control. Obviously the V_{DDL} of first layer is *0.5V* while the second layer is *0.7V*. The formulations are as follows and we can tell that the stage-level is more efficient. In this case, the power reduces by $(4.43 - 3.47)/4.43 = 21.6\%$. Formulations are as follows:

$$\min(0.7^2 \times 4 + 0.7^2 \times 3 + 1^2, 0.5^2 \times 4 + 1^2 \times 4) = 4.43$$
$$0.5^2 \times 4 + 0.7^2 \times 3 + 1^2 = 3.47$$

Secondly, we demonstrate a sample in Fig. 3 b to prove that different Free Node Arrangement could deduce great power consumption difference under equal *II* scheduler. In this case, we always apply the stage-level granularity voltage control. Each stage has only two supply voltage, the $V_{DDH} = 1V$ and its unique V_{DDL}. There is no possible to have two V_{DDL}s.

In Fig. 3 b, a 5 operands DFG with two free nodes can achieve 3 different mapping patterns all with the same $II = 2$. We can calculate the power cost left to right, top to bottom using above simplified method and represent them as follows. Right bottom is the best one and it reduces the power by $(2.96 - 2.48)/2.96 = 16.2\%$ compared to the worst choice.

$$(1^2 + 0.5^2) + \min(1^2 \times 2 + 0.5^2, 0.7^2 \times 3) = 2.72$$
$$1^2 + \min(1^2 \times 2 + 0.5^2 \times 2, 0.7^2 \times 4) = 2.96$$
$$(1^2 + 0.5^2 \times 2) + 0.7^2 \times 2 = 2.48$$

We can get a hint that different mapping patterns with equal *II* can have different optimal V_{DDL} pairs hence can generate dramatic power consumption difference. However, Graph Minor focuses only on how to reduce the *II*, neglecting dramatic power consumption difference between two equal *II* schedules with different patterns. We improve Graph Minor by adding cost function concerning power consumption and node arrangement guide. We once tried three different granularities:

i) Global granularity voltage optimization method. It's a direct transplantation from Zhu's Dual-V_{DD} simulation method to loop mapping. Though it can reduce the power reduction, it does not fully utilize the feature of loop mapping application.

ii) Appealing to stage-level granularity voltage control. We find that each stage of kernel usually has different operands distribution hence can has unique optimal V_{DDL} rather than the global equal V_{DDL} in Zhu's brief. In Fig. 3, we have an insight that this modification is powerful. We have reasons to believe that this further considering would reduce power consumption. Though more compilation time is the cost, result shows the cost is acceptable.

iii) For logical completeness and with the ii) method performance inspire, we hope to take full use of the flexibility. We had a trial on assigning unique optimal V_{DDL} to each PE according to its own operand types. But we quickly abandon this expansion for its unaffordable additional power cost of Level Shifter for transmitting signals among diversified VDDs, which counteracts the potential power income as our experiments shows. Thus, we accept the method ii) for its higher efficiency at the cost of a tolerated additional compilation time.

4. Implementation

This section details math formulations of estimating the optimal V_{DDL} and describes how to wrap them into a cost function by which we can improve original pruning constraints of Minor Graph. We should export a datasheet of feasible region of V_{DDL} variation of ALU, Interconnect, Level Shifter through experiment. Then we provide a high-level view of algorithm and a detailed *ReMinor* function.

4.1 Optimal V_{DDL} Estimation

$$P = \sum_i \left[P_A(i) + P_I(i) \times \left(\frac{V_{(i)}}{V_{DDH}} \right)^2 + P_{LS}(i) \right] \quad (1)$$

$$D_A(i) + D_I(i) \times u(i, V_{DDL}) \leq D_C \quad (2)$$

$$u(i, V_{DDL}) = \frac{t_p(V_{(i)})}{t_p(V_{DDH})} \quad (3)$$

$$V_{(i)} \in \{V_{DDL}, V_{DDH}\} \quad (4)$$

$$i \in PEs \times PEs \quad (5)$$

P_A, P_I, and P_{LS} are the power of the ALU, interconnect and level shifter. D_A is the delay of the ALU in each PE. D_I is the delay of interconnect. D_C is the critical path delay (e.g. multiplier). $u(i, V_{DDL})$ is the delay which is a function of voltage variation. $V(i)$ is the supply voltage of the interconnect. The number i represent individual PE.

The whole execution time is affected by the voltage variation. However, power consumption of ALU and level shifter can be seen as constant in various voltage, thus above equations can be simplified accordingly.

We pack up the whole equations to a power cost function *PV* by which we can calculate the optimal V_{DDL} for each stage of kernel if we set our goal to make a minimum P in Eq. (1). If we stop our exploration and are satisfied with the combination between Graph Minor and Variable Dual-V_{DD}, we are still going to reduce power consumption sharply. In this situation, we firstly apply Graph Minor method to get a valid scheduler and appeal to the *PV* to calculate the optimal V_{DDL}. After that we modify the scheduler by writing the V_{DDL} information to our *II* different configurations. Experiment shows the average reduction ratio is 22.2%. We want to incorporate them into an intact method and utilize the full flexibility, so we develop Revised Graph Minor function as follows.

4.2 High-Level View of Algorithm

Mapping loop kernel onto CGRA is equivalent to finding a valid graph *M*, which is a subgraph of the MRRG, such that the original DFG can be obtained through edge contractions from *M*.

In the preprocessing stage, ReMinor changes input DFG, denoted with H, to satisfy the necessary mapping conditions such as outdegree constraints and level constraints (line 2, line4). Then it calculates the ideal *MII* (minimum *II*) by existing solution introduced by traditional modulo scheduling of VLIW processor (line 5). For efficient mapping, it chooses an appropriate order of nodes of H. We could map the DFG much quickly and efficiently with the help of this ordering (line 6). Then it fulfills the mapping process in the *while* loop. An SRG (schedule and route graph), which contains the scheduling and routing information, is created from MRRG corresponding to the CGRA features. The height of SRG is equal to our testing *II*, which

Algorithm1 Revised Graph Minor Mapping Algorithm

```
1: Begin
2:     H_p ← Constraint_Outdegree(H);
3:     H_p ← Constraint_balance(H_p);
4:     H_p ← Constraint_Level(H_p);
5:     II  ← Max(resMII, recMII);
6:     list ← Ordering(H_p);
7:     while mapping is not found do
8:         C_II ← Creat_MRRG(C, II);
9:         if ReMinor(H_p, C_II, M) then
10:            Return(M)
11:        else II ← II + 1
12:        end if
13:    end while
14: End
```

is also supposed to be the number of configurations of a successful scheduler (line 8). In the next stage, we attempt to find a valid mapping through Graph Minor Test, which is denoted as function *ReMinor(H_p C_{II} M)*. The graph *M* is defined as our final successful mapping pattern graph (line 10). If the H_p is the minor of C_{II}, the DFG can be mapped with the current initiation interval *II*. Applying Graph Minor Theory, we know the H_p is a minor of C_{II} if there is a graph *M* which is subgraph of C_{II}. After all the attempts, we either return a successful mapping graph *M* or increase *II* to initial a new attempt (line 9-11). At last, we get a successful mapping or prove that there is no valid mapping for this DFG.

Chen's core program takes validity into account only, where all the possible mapping between DFG and MRRG are executed in *Minor(H, C_{II}, M)*; thus this process does not omit some invalid mapping choices for there is no preprocessing technology. Obviously, the searching space is very large, which is exponential to the number of nodes of DFG. For reality application, we should reduce the searching space by our preprocessing constraints and some pruning constraints. In addition to the core testing, *ReMinor(H_p C_{II} M)* capsule some heuristics which is designed to accelerate the mapping process. These heuristics avoids further attempts if it foresees this mapping is supposed to be failed. Now we introduce our preprocessing constraints and core pruning constraints which are carefully designed and added into Chen's original algorithm.

4.2.1 Degree and Level Number Constraints

Chen's original algorithm does offer a degree constraint checker but he neglects to fix some topology constraints of the DFG. Firstly, the out-degree of every operation must be less than the out-degree of PEs in the MRRG. Without this

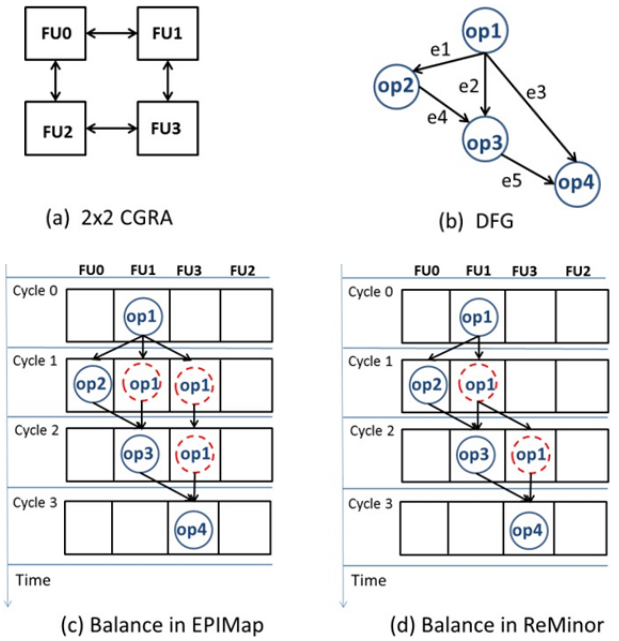

Fig. 4 Balance constraints in two methods.

constraint, a feasible mapping cannot be found. A proper solution to fix the problem might be the *re-computation* mentioned in EPIMap. Secondly, the number of nodes in each level of DFG must be less than the number of PEs in CGRA (non–time extended). If the number of nodes in a level exceeds the capacity of CGRA, we need to move the exceeded ones to previous or next levels during our mapping tests, which would consume too much time. It is wise if we satisfy the level number constraints using *re-computation* technique before the mapping. Graph H in Chen's algorithm is as exactly the same graph as DFG, while H_p in ours is an epimorphic version of original DFG.

4.2.2 Balance Constraints

Balanced DFG in ReMinor obtained through *routing* is little different from that in EPIMap. In EPIMap, the route PE cannot be shared with two different edges, while route sharing is the biggest feature in Graph Minor.

As shown in Fig. 4, the arc *e2* is not balanced because the node *op1* (scheduled at cycle 0) and *op3* (scheduled at cycle 2) is mapped on CGRA where op3 cannot receive the data-out from *op1* if there is no replica of *op1* in cycle 1. The edge *e3* is the same as *e2*. In order to get valid mapping, we must make the original DFG balanced and execute the revised DFG as shown in Fig. 4-d. There is no route sharing.

Chen's original Minor Mapping doesn't introduce balance constraints. There is a function called *expand()* which would replicate *op1* in (FU1, cycle 1) and (FU3, cycle2) when it finds all the attempts have failed. The final pattern is shown in Fig. 4-d.

However, the function *expand()* is time consuming. Inspired by balance technology of EPIMap, we add balance constraints to Graph Minor. In addition, the number of replicate PEs is smaller than that in EPIMap with the help of shared routes feature of Graph Minor. Revised DFG is shown as the new DFG in Fig. 4-d.

Beware of the difference between the two balance constraints, the revision in EPIMap is necessary to obtain a valid schedule while that of ReMinor is just an acceleration strategy.

4.2.3 DFG Node Ordering

The nodes along the critical path have higher priority because they appear earlier than others. If this critical path cannot be mapped successful under current *II* value, we can abort immediately without wasting time and move on to the next *II*. A node v can be mapped only when at least one of its direct predecessor or successor has been mapped.

Ordering process can benefit from *re-computation* or *routing*. Some DFG cases, which fail to map after all the iterative attempting, can be reshaped through replicating some nodes in critical path such that we can get a successful mapping. In these situations, the *list* should be updated according to the details of *re-computation* or *routing*.

However, we dislike replicate of nodes in critical path because it may cost large extra energy consumption, which is contrary to our goal of minimizing power consumption. So we do not use this technique until we fail using normal critical path order. We never update *list* in *while* loops.

We have no choice but to do *re-computation* to get a valid mapping in the former case, while we might successful map our DFG to CGRA without updating critical path order. Hence the cost of *re-computation* in degree constraints is a necessity which we much bear.

4.2.4 Attribute and Other Constraints

In addition to describe the functionality of specified PE, for example, memory-only, multiply-only or omnipotent except memory function, we must focus on how to calculate power consumption of different V_{DD}. Thus we add *V_{DD}-Execution Time* relationship, which is exported from our simulation experiment, to our attribute table. Only after this modifying can we incorporate power estimation into the placement and routing process. Hence the meanings of available resources are extended from concerning the functionality only to both the power consumption and functionality. The data dependence and timing constraints are satisfied as the original algorithm suggests.

If we map a path r, from node p to node n, of graph H to a path R of graph M, we must guarantee the path delay of R is no smaller than that of r. If node n does not have any mapped direct successor, the corresponding node sets in graph M can safely be set to one single node. Otherwise, the corresponding nodes are more than one in which the additional nodes are used as route function to ensure data dependence and timing constraints.

Algorithm2 ReMinor(H_p,C_{II},M)

```
1:  Begin
2:      if no unmapped node in H_p then
3:          if return success then
4:              PowerCost ← PV(M)
5:              PowerCost ← update all the valid pattern saved in δ
6:              return the smallest one
7:          else if return fail then
8:              Replicate and reordering critical path
9:              Reattempt until II is larger than critical length
10:         end if
11:     else the next node β in H_p
12:         P(β) ← all mapped direct predecessors of β
13:         S(β) ← all mapped direct successors of β
14:         φ(β) ← min_map( β, P, S)
15:         φ(β) ← Pred(φ(β))
16:             if two euqal lenghts of path then
17:                 φ(β) ← randomly chose one
18:                 θ ← the unchosen pattern of β
19:                 δ ← Pred(θ)
20:             else if recomputation exist then
21:                 φ(β) ← chose recomputation
22:                 θ ← the original pattern of β
23:                 δ ← Pred(θ)
24:             end if
25:         M ← M ∪ φ(β)
26:         if Minor(H_p, C_II, M) == 1 then sucess
27:         else φ(β) ← expand_map( β, P, S) goto min_map
28:         end if
29:     end if
30: End
```

4.3 Function *ReMinor*(H_p, C_{II}, M)

Original Graph Minor Framework concerns *II* only and ignores the pattern difference. Hence the output is only one schedule though there may be multiple valid patterns with the same *II*. Our main purpose is to find the minimum power one from these valid patterns, for they may have quite different power consumption. The biggest evolution of *ReMinor* is to generate all the valid patterns with smallest *II* and choose the minimum power one (line3-6). To obtain all the patterns, we should save the breakpoint information of mapping when we face two choices with equal time and possible different power consumption simultaneous. For pruning the search space, we also transplant the prediction technology of GMinor to look ahead in future and quickly define if the current alternative can be extended to successful mapping. It is abandoned if the *Pred* function returns false (line 16-23). *Pred* also find its use in formal plan (line 15).

The reordering process (line7-9) has been discussed in 4.2.3. The third contribution is the PV function and *re-computation* technology.

The lesser number of nodes and edges, the smaller execution time of mapped DFG. For reducing power consumption without decreasing performance, we adopt *min_map* heuristic by which we find minimal mapping pattern,

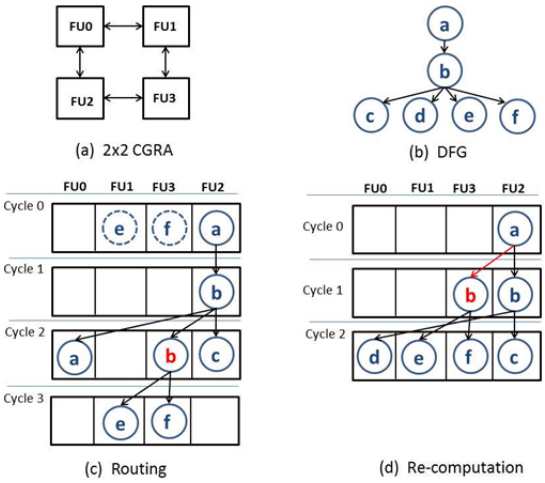

Fig. 5 Re-computation v.s. Routing.

containing minimal number of nodes and satisfying all the constraints, from each original edge to the corresponding edge in M. Then we do Graph Minor check in function $Minor(H_p, C_{II}, M)$, which ensures the shared routes feature and the validity of the mapping. All the direct predecessors/successors of v are connected with $\emptyset(v)$ in a shortest path. *expand* is designed to solve the failure of *min_map* where we add one extra node to fulfill the routing path.

While old method always chose *routing* in *min_map* and *expand*, *re-computation* introduced in ours will be useful to reduce power consumption without affecting performance. Firstly, the length of mapped path of *re-computation* is always shorter than that of *routing*, which will meet our minimal mapping constraints. Considering node b to node f in Fig. 5, the length of shortest path using *re-computation* is 1, while that number of *routing* is 2. Secondly, we can achieve power reduction using *re-computation*. Additional number of PEs, which satisfies the necessary constraints, is exactly the same in both *re-computation* and *routing*. Though additional PEs in *routing* consume less energy for they need not to execute again, *re-computation* can reduce the power consumption as a whole.

For example, *a* is *adder* operation while others are *not* operations in Fig. 5. The minimum permissible V_{DDL} is *0.7V* and *0.5V* respectively in our assumption. PE for route only can be assigned to 0.5V because the delay is just like that of *not* operation. Observing the kernel pattern difference between the two, only node f, e can have different optimal minimum V_{DDL}.

In *re-computation* all the nodes can execute in their minimum permissible V_{DDL}, the power consumption: $0.7^2 + 0.5^2 \times 6 = 1.99$. In *routing*, node f, e execute in *0.7V* other than its minimum permissible V_{DDL} because in first layer: $0.5^2 \times 2 + 1 = 1.5 > 0.7^2 \times 3 = 1.47$. Power consumption of whole kernel is: $0.7^2 \times 3 + 0.5^2 \times 4 = 2.47$. The reduction ratio is $(2.47 - 1.99)/2.47 = 19.4\%$.

5. Results

Our experiment platform is assumed to be: 4 x 4 PEA, 9-kbit data FIFO with 256 bits width, 160-kB configuration memories whose data width is 1024bits. It is synthesized at 100MHz with Synopsys Design Compiler and TSMC 65-nm library. The execution time of different operations is exported through static timing analysis. The normalized ALU delays under different configurations, i.e. different operations, are given in Table 1. The permissible minimum supply voltage of different operations is shown in Table 2 which is obtained through simulation method. The data of both tables is critical to cost function *PV*.

We select nine loops from benchmark programs in MiBench and SPEC2006 whose characteristics are listed in Table 3. To inherit and compare with the former research contributions, we assume our 4 x 4 PEA is homogeneous, and PEs is capable of handling fixed-point and logical operations in any type.

5.1 Results of Three ReMinors

We find that the stage-level is efficient than other two granularity. The average reduction ratio of three is shown in the rightmost column of Fig. 6 where the ratios of *global*, *stage* and *pe-level* are 17.4%, 27.6% and 4.5% respectively. There is a $(27.6 - 17.4)/17.4 = 58.6\%$ upgrade when we utilize stage-level instead of previous global optimized V_{DDL}.

In most test benches, apart from *wavelet* and *sobel*, the *stage-level* performs much better than *global-level*. Differences in the two exceptional benches are negligible, so we can safely expect that the *stage-level* is not only a wiser choice in average sense but also a reliable method to various applications.

Level-Shifter and fine-granularity control hardware counteract income for its managing cost, especially in the situation of large DFG, such as *fft*, *idctfft* whose nodes are more than 40, where we gain nothing from this fine granularity. *PE-level* is unacceptable as the result shows for its additional overhead.

We always choose the *stage-level* as the results proves.

Table 1 Normalized delay.

OP code	Delay	OP code	Delay
*	0.74	>/>=	0.60
+/-	0.65	</<=	0.53
==	0.55	!=	0.57
<<	0.65	load/store	1.00
and/or_d	0.38	and/or_s	0.47
not_d	0.12	not_s	0.20
xor	0.50	register	0.10

Table 2 Minimum permissible supply voltage.

OP code	Delay	OP code	Delay
*	1.00	>/>=	0.85
+/-	0.85	</<=	0.75
==	0.80	!=	0.80
<<	0.85	load/store	1.00
and/or_d	0.60	and/or_s	0.70
not_d	0.55	not_s	0.50
xor	0.70	register	0.50

Table 3 Characteristics.

Benchmark	#ops	#MEM	#edges
wavelet	12	4	6
scissor	12	4	13
osmesa	16	9	17
sor	17	6	11
lowpass	23	9	34
sobel	27	7	34
fft	40	20	42
tiff2bw	42	20	50
Idctflt	87	25	117

5.2 Comparison between *ReMinors* and *Direct*

Firstly, we compare the *direct method* with the *global* granularity ReMinor. Depicted in Chapter 4, the *direct method* is a simple combination between Graph Minor and voltage control where we directly accommodate the supply voltage of configurations obtained by Graph Minor Mapping Process. There is no guide information from PV function. The ratios of *direct* and *global* are 17.4% and 22.2% respectively. Though *wavelet*, *osmesa* and *sobel* is more suitable of *global ReMinor*, the gain compared to *direct* is negligible and the ratios of other cases in *direct method* are much higher than that of *global ReMinor*.

Secondly, the *direct method* is revised to *stage-level* ReMinor and we elevate the ratio from 22.2% to 27.6% on average. And the *stage-level* ReMinor is always efficient than direct method for all the test benches due to the essence of ReMinor algorithm.

5.3 Reasonable Compilation Time

EPIMap and GMinor [11] have been well accepted as the best known efficient accelerating strategies, therefore the compilation time of both methods is reasonable which enables us to compare the compilation of ReMinor with that

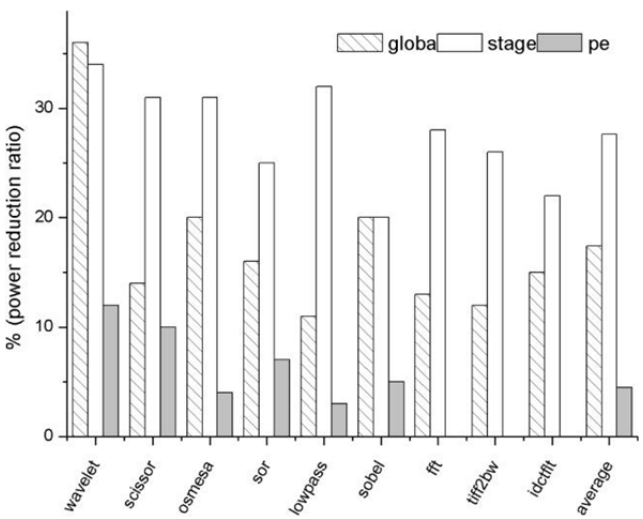

Fig. 6 Reduction ratios of ReMinor.

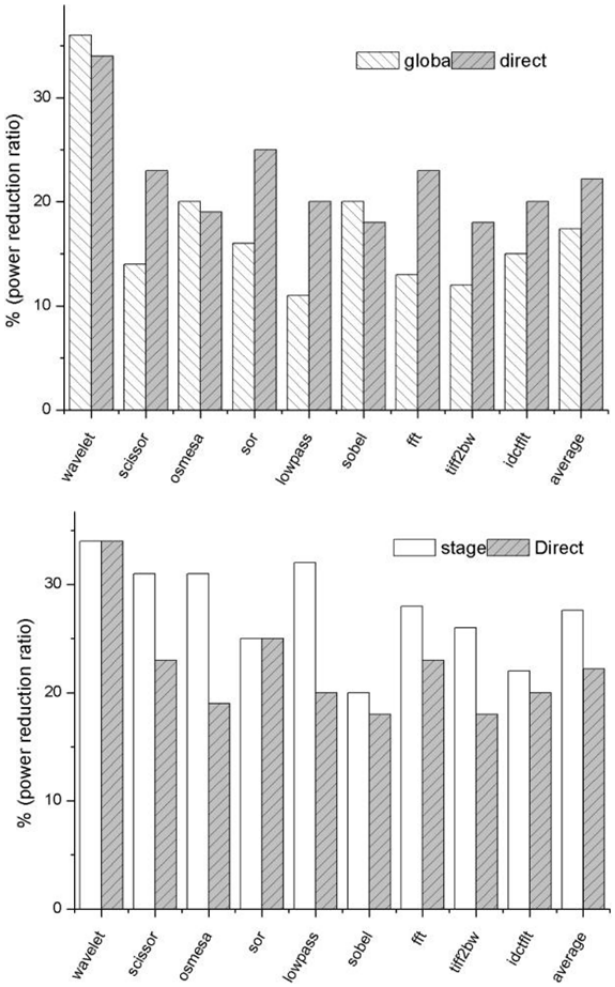

Fig. 7 Comparison between ReMinor and direct.

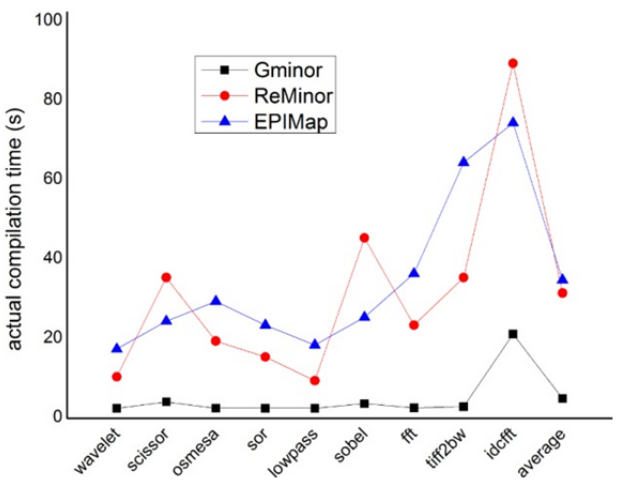

Fig. 8 Compilation time analysis.

of EPIMap and GMinor to check out whether the compilation of ReMinor is acceptable.

As shown in Fig. 8, the average compilation time of GMinor is 2.49 sec, which is slightly smaller than the 3.4 sec reported in [11] because of the different test benches selected, while that of EPIMap is 32.4 sec, which is consistent with the timing reported in [7]. The huge difference between the two number is that EPIMap has to repeat MCS kernel computation when the mapping fails, i.e. has to endure many iterative steps. ReMinor is derived from GMinor by adding some constraints and iterative processing, therefore the compilation time is much more than that of GMinor. The number is 29.1 sec shown in Fig. 8. The additional compilation time is the cost of pattern save and selection for power reducing.

Furthermore, we extract 50 random DFGs with nodes distributed in the range $(0, 100]$ to check out the compilation time under the three methods. The result of GMinor, EPIMap and ReMinor is 4.3 sec, 52.6 sec and 47.3 sec respectively.

Since the compilation of ReMinor is smaller than that of EPIMap, we testify that the whole compilation time of ReMinor is acceptable.

6. Conclusion

Utilizing slack time through Variable Dual-V_{DD} architecture to reduce power consumption is not only depended on the utility ratio of the idle time but also affected by the additional cost of hardware and scheduling. By trade-off, stage-level granularity voltage control is chosen to combine with graph minor loop mapping method. If we only simply do the Graph Minor Mapping and then apply stage-level voltage control in each stage of kernel, we still can get an average power reduction ratio by 22.2%. In this brief, we explore further and finally formulate Revised Graph Minor Mapping Algorithm which incorporates stage-level voltage control to loop mapping domain. Our method reduces power consumption by 27.6% on average without compromising performance.

References

[1] R.B. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," Proc. MICRO, pp.63–74, 1994.
[2] B. Mei, S. Vernalde, D. Verkest, H.D. Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," Proc. Field Programmable Logic and Application, 13th International Conference, pp.61–70, Sept. 2003.
[3] Y. Kim, I. Park, K. Choi, and Y. Paek, "Power-conscious configuration cache structure and code mapping for coarse-grained reconfigurable architecture," Proc. 2006 International Symposium on Low Power Electronics and Design, pp.310–315, Oct. 2006.
[4] H. Park, K. Fan, M. Kudlur, and S. Mahlke, "Modulo graph embedding: Mapping applications onto coarse-grained reconfigurable architectures," Proc. 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pp.136–146, Oct. 2006.
[5] T. Schweizer, T. Oppold, J.O. Filho, S. Eisenhardt, K. Blocher, and W. Rosenstiel, "Exploiting slack time in dynamically reconfigurable processor architectures," Proc. Int. Conf. Field-Programm. Technol., pp.381–384, Dec. 2007.
[6] H. Park, K. Fan, S. Mahlke, T. Oh, and H. Kim, "Edge-centric

modulo scheduling for coarse-grained reconfigurable architectures," Proc. PACT, pp.166–176, 2008.
[7] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," Proc. 49th Annual Design Automation Conference (DAC), pp.1284–1291, June 2012.
[8] J. Zhu, L. Liu, S. Yin, and S. Wei, "Low-power reconfigrable processor utilizing variable dual V_{dd}," IEEE Trans. Circuits Syst. II, Express Briefs, vol.60, no.4, pp.217–221, April 2013.
[9] D. Liu, S. Yin, L. Liu, and S. Wei, "Polyhedral model based mapping optimization of loop nests for CGRAs," Proc. 50th Annual Design Automation Conference (DAC), Article No.19, June 2013.
[10] J. Zhu, L. Pan, Y. Yan, D. Wu, and H. He, "A fast application-based supply voltage optimization method for dual voltage FPGA," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., pp.2629–2634, 2013.
[11] L. Chen and T. Mitra, "Graph minor approach for application mapping on CGRAs," International Conference on Field Programmable Technology (FPT), pp.285–292, 2012.

Bing Xu received his B.S. degree from the Department of Information Science and Technology, Shandong University, China, in 2009. Currently he is pursuing the M.S. degree in the Institute of Microelectronics, Tsinghua University, Beijing, China. His research interests include reconfigurable computing and its compiler design.

Shouyi Yin received the B.S., M.S. and Ph.D. degree in Electronic Engineering from Tsinghua University, China, in 2000, 2002 and 2005 respectively. He has worked in Imperial College London as a research associate. Currently, he is with Institute of Microelectronics at Tsinghua University as an associate professor. His research interests include mobile computing, wireless communications and SoC design.

Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree in Institute of Microelectronics, Tsinghua University, in 2004. He now serves as an associate professor in Institute of Microelectronics, Tsinghua University. His research interests include reconfigurable computing, mobile computing and VLSI DSP.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.