Register Allocation for VLIW DSP Processors with Irregular Register Files *

Yung-Chia Lin, Yi-Ping You, and Jenq Kuen Lee

National Tsing Hua University
Hsinchu 30013, Taiwan

Abstract. A variety of new register file architectures have been developed for embedded processors in recent years, promoting hardware design to achieve low-power dissipation and reduced die size over traditional unified register file structures. This paper presents a novel register allocation scheme for a clustered VLIW DSP processor which is designed with distinctively banked register files in which port access is highly restricted. With the specific register file organizations considered to decrease the power consumption because of fewer port connections, not only does the clustered design make register access across clusters an additional issue, but the switched access nature of the register file demands further investigations into optimizing register assignment for increasing instruction level parallelism. We propose a heuristic algorithm to obtain preferable register allocation that is expected to well utilize the irregular register file architectures. Experiments were done with a developing compiler based on the Open Research Compiler (ORC), and the results showed that the compilation with the proposed approach delivering significant performance improvement, comparable to a simulated annealing approach which is considered not as a near-optimal but an exhaustive solution.

1 Introduction

The high computing power required by today's numerous embedded applications, which cannot be sufficed by typical RISC embedded processors and low-end DSP with little parallelism, continuously propels the investigation into the efficient parallel-architectures of DSP. In the design of DSP for embedded processing with greater parallelism, power consumption and chip die size are always to be the significant concerns on top of higher processor performance. Exploiting instruction-level-parallelism by VLIW architectures is the current trend for designing a high-performance DSP processor, which usually requires a large number of registers in a unified file to optimize resource utilization in the instruction scheduling with minimizing slow memory traffic. Such a common approach for register file design, like Intel's Itanium VLIW processors, is not feasible for

* The work was supported in part by NSC under grant no. 94-2220-E-007-019 and 94-2 220-E-007-020, by Ministry of Economic Affairs under grant no. 94-EC-17-A-01-S1-034, and by MOE research excellent project under grant no. 94-2752-E-007-004-PAE in Taiwan.

embedded DSP processors due to the design constraints of power dissipation and chip die size. Furthermore, under the consideration of the circuit design, larger number of registers that are accessible by all functional units demand more number of ports which intensely hinder the access time for such a register file, restricting the possible processor cycle duration and adding the difficulty of the design [4]. To solve this weakness, a variety of decentralized register file architectures have been developed for embedded processors in recent years, promoting hardware design to achieve low power dissipation and reduced die size over traditional unified register file structures.

One of the techniques to decentralize a unified register file is employing clustering concept to partition register files for different groups of functional units. For example, Texas Instruments TMSC6x DSP [14] series use homogeneous clustered architectures with partitioned register banks and CEVA's CEVA-X [2] architectures utilize heterogeneous clustered architectures with partitioned register files. Another technique to improve power dissipation without potentially performance degradation is using an irregular access restricted design of register file structures, such as windowed register files [12] and hierarchical register files [13]. Unfortunately, due to the more specific accessing features and irregular constraints that are usually held by processors incorporating such partitioned register file organizations, more appropriate code generation, register allocation, and instruction scheduling schemes than conventional compilation techniques are in great demand to attain optimal performance.

This paper describes a novel register allocation scheme for a clustered VLIW DSP processor, known as Parallel Architecture Core (PAC) DSP [3,9,10], which is designed with distinctively banked register files in which port access is highly restricted. The PAC DSP employs a heterogeneous design that equips one singular scalar unit (for light-weight arithmetic, address calculation, and program flow control), plus two data stream processing clusters in which each one contains a pair of load/store unit and ALU/MAC unit with powerful SIMD capabilities; every unit in the clusters collocates three varied types of register files, providing different accessing manners and constraints, and the scalar unit has its own accessible register file deployed. The major specialty of the register file architectures featured by the PAC DSP processor is that it incorporates a so-called *ping-pong register file structure* [8], which is divided into two banks and in which banks can only be restrictedly accessible in a mutual-exclusive way, as a semi-centralized register file among clusters and functional units within a cluster. With this design to decrease the power consumption because of fewer port connections, not only does the clustered design make register access across clusters an additional issue, but the switched access nature of the ping-pong register file raises our interest in investigating further register assignment to increase instruction level parallelism.

We propose a heuristic algorithm, named as *Ping-pong Aware Local Favorable* (PALF) register allocation, to obtain preferable register allocation that is expected to well utilize the irregular register file architectures in PAC DSP. The algorithm involves the proper consideration of various characteristics in ac-

cessing different register files, and attempts to minimize the penalty caused by the interference of register allocation and instruction scheduling, with retaining desirable parallelism over ping-pong register constraints and inter-cluster overheads. Experiments were done with a developing compiler for the PAC DSP based on the Open Research Compiler (ORC), and the results indicate that the compilation with the proposed approach delivers significant performance improvement, comparable to a simulated annealing approach which is considered not as a near-optimal but an exhaustive solution.

The sections of this paper are organized as follows. In section 2 we will introduce the processor architecture and register file organizations of PAC VLIW DSP. Section 3 will brief the complicated issues caused by the severe correlation between code generation, register allocation, and instruction scheduling in PAC architectures. The proposed PALF register allocation scheme will be addressed in section 4 which is followed by an illustrative sample in section 5. The experimental results of our evaluation and related discussion are detailed in section 6. Finally, section 7 concludes this paper.

2 Ping-pong Register Files with Clustered Architectures

This section overviews the PAC DSP VLIW architecture and its irregular register file design.

2.1 PAC DSP Architectures

The PAC DSP originally features a clustered VLIW architecture which boosts scalability, and a large number of registers which are arranged as innovative heterogeneous and distinct partitioned register file structures. Being unlike symmetric architectures of most DSP processors available nowadays, the PAC DSP processor is constructed as a heterogeneous five-way issue VLIW architecture, comprised of two integer ALUs (I-unit), two memory load/store units (M-unit), and the program sequence control unit/scalar unit (B-unit) which is mainly in charge of control flow instructions like branch and jump. Each unit has its own executable subset of instruction set and each executable instruction has its own register accessibility and constraints. The M- and I-units are organized in pairs, and each pair contains exactly one M-unit and one I-unit to form a cluster arrangement with associated register files. It is apparent that each cluster is logically appropriate for one data stream processing, and the current design of PAC DSP consists of two clusters to support maximum workload capacity of two concurrent data stream. But the scalability of the cluster design in PAC DSP could allow the processor to easily involve more clusters to handle larger data processing workload demand. The B-unit consists of two sub-components, the program sequence control unit, and the scalar unit, due to the hierarchical decoder design for variable-length instruction encoding in PAC DSP. The program sequence control unit primarily takes charge of operations of control flow instructions. The scalar unit, which is capable of simple load/store and address

arithmetic, is placed separately from data stream processing clusters, with its own register file. The overall architecture is illustrated in Fig. 1.

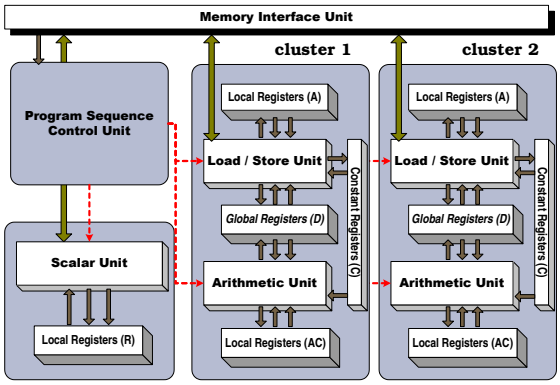

Fig. 1. The PAC DSP architecture illustration

2.2 Irregular Register Files and Access Constraints

As shown in Fig. 1, registers in PAC DSP are organized into several distinct partitioned register files and placed as cluster structures, to reduce wire connections between functional units and registers so that chip area and power consumption may be decreased. All units in the processor have their dedicated local register files attached: they include R-register file, AC-register file, and A-register file, which are only accessible by B-unit, I-unit, and M-unit, respectively. In each cluster comprised of M-unit and I-unit, a global register file named as D-register file is designed to be shared by the pair of M- and I-units in each cluster. The internal of the D register file is further partitioned into two banks to utilize the instructional port switching technology in order to reduce more wire connections between the M- and I-units. This technology, being referred to the name as '*ping-pong register file structure*', is that decreasing the register bank port connection which limits the accessibility of the two bank; in each cycle, the two functional units can only access to different banks. Each instruction bundle encodes the information of which bank is to be accessed for each functional unit in the cycle so that the hardware can do port switching between D-register file banks and functional units, to attain the purpose of data sharing within a cluster. By using the concept of overlapping two different data-stream operations in a cluster, we may minimize the occasion that M- and I-units access the same data at the same time; therefore, the access constraints of '*ping-pong register file structure*' should cause little impact on performance. The advantage of such a '*ping-pong register file structure*' design is believed to consume less power due to its reduced read/write ports [13] while retaining the data communication capability. Figure 2 illustrates the constraints of the ping-pong register file. Besides local register files and global register files, each cluster contains an additional

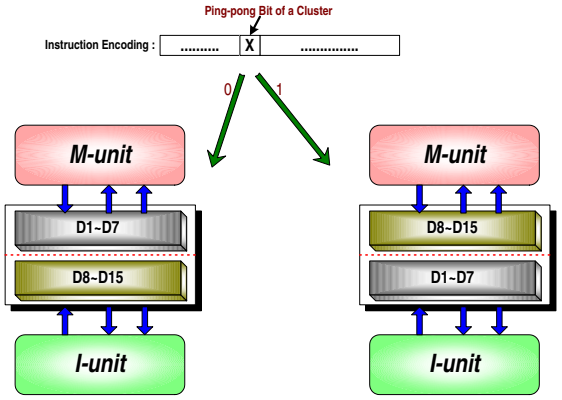

Fig. 2. The access constraint of the ping-pong register file structure

constant register file which is shared by both M- and I-units as one of the read-only operand sources usable by certain instructions. Only M-units can initialize the data in the constant register file.

3 Optimizing Allocating with Irregularity

For architectures supporting instruction-level-parallelism, effective register allocation with scheduling is always one of the crucial issues to optimize code performance. Register allocation and instruction scheduling are typically done in separate phases for most compiler infrastructures to decrease the complexity of these two combinatorial optimization problems. For separate orders of processing register allocation and instruction scheduling, interference between these two processes should be considered to determine a suitable performing sequence based on the architectural features of the target machine. If register allocation is done after instruction scheduling, we may always get unfeasible register allocation for a certain schedule, particularly on architectures with heterogeneous design and irregular constraints. Therefore, performing register allocation before instruction scheduling is more favorable for PAC architectures in our compiler development than other target machines. Since register allocation may create additional dependencies and restrictions that impacts the later scheduling due to the register usage and liveness, register allocation is required to be optimized so that the instruction-level-parallelism could still be achieved after the allocation.

Comparing to other platforms, PAC architectures introduce severe issues to be dealt with register allocation. First, the access constraints of ping-pong register file structures restrict the scheduling of two instructions that use the global D-register files in the same cluster in a cycle no matter if they have dependencies or not. Second, inserting additional data communication code into the original program will frequently be required while exploiting instruction-level-paralllelism because of the highly-partitioned register files and their accessibility. A pair of explicit instructions must be issued together, for instance, to transfer data from one cluster to another cluster, which use the internal routing

data-path of the memory interface unit and occupy two slots in an instruction bundle. Both of the issuing slots occupation and execution latency will affect the scheduling of the two-cluster programs that need inter-cluster communication. Intra-cluster data communication code insertion is also a common case after the

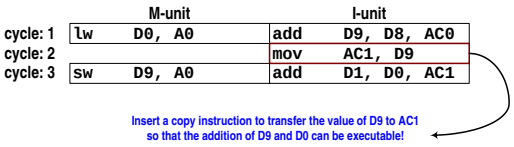

Fig. 3. An example of intra-cluster communication insertion

register allocation that utilizes more register files to optimize the scheduling, like the example shown in Fig. 3. To properly handle these issues with register allocation, we propose a heuristic approach to adapt general graph coloring techniques for each register files and attain a profitable solution for PAC compilers.

4 Ping-pong Aware Local Favorable Register Allocation

In this section we present a register allocation algorithm which, given a dependency DAG (Directed Acyclic Graph) [1] that describes the compilation regions, heuristically determines the proper register file/bank assignment and employ state-of-the-art graph-coloring register allocation for each assigned register file/bank in PAC architectures.

4.1 Overview

The overall register allocation algorithm proposed is shown in Fig. 4. Our ap-

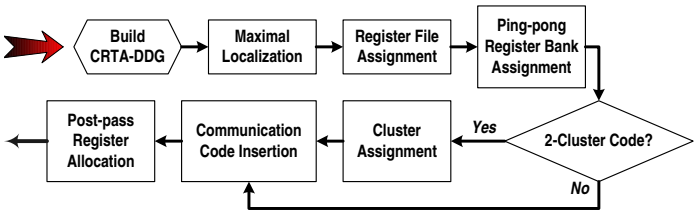

Fig. 4. The flowchart of LFRA scheme

proach requires building an extended data dependence DAG, called the Component/Register Type Associated Data Dependence Graph (CRTA-DDG), which preserves the information of the execution and storage relationship for irregular constraint analysis, in addition to the original partial order imposed by instruction precedence constraints. Nodes in a CRTA-DDG represent instructions of

the input code block, with the component-type association (that indicates which functional unit is preferred to be scheduled for this node) and the register-type association (that annotates the appreciated physical register file/bank, to where the operands/results will be allocated); the edges linked between the nodes represent data dependency that serializes the execution order to be followed in the scheduled code sequence. An example code sequence and its CRTA-DDG are shown in Fig. 5, where a TN (TemperaryName) of register type is referred as a virtual register required to be allocated to a physical register in the machine-level IR used by ORC. The advantage of using CRTA-DDG is that it clarifies

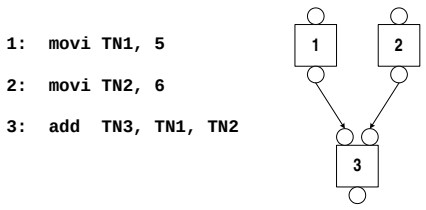

Fig. 5. A simple code with its CTRA-DDG

the allocation and schedule restrictions for each node with the consideration on complex constraints in PAC architectures, while the well-developed graph partitioning methods may still easily be applied to our register allocation algorithms. The main PALF register allocation scheme could be organized into five phases as follows:

1. Build the CRTA-DDG and perform the preferable functional unit assignment to the default execution type of each instruction by the "maximal localization" analysis.
2. Assign operands/results (required to be allocated to physical registers) of each node in the CRTA-DDG to the desirable register files.
3. Partition the operands/results assigned to the global ping-pong register files to the preferred register banks by the strategy of optimizing ping-pong parallelism.
4. Partition the nodes in the CRTA-DDG into two clusters properly if the "compiling for two-clusters" option is set.
5. Insert nodes of required communication code to avoid invalidity caused by the register file/bank assignment and cluster-partitioning, followed by the physical register allocation for each register file.

4.2 Maximal Localization

Assume a set of v nodes $V = \{n_1.n_2, \ldots, n_v\}$ with r TNs $R = \{t_1, t_2, \ldots, t_r\}$ are in a given CRTA-DDG, $G = (V, R, E)$, and the dependencies of these nodes are represented by e directed edges, each of which is denoted by (t_i, t_j), whereas $1 \leq i \leq r$, $1 \leq j \leq r$, $t_i \in R$ and $t_j \in R$. An operand or result referred by an instruction is represented as a TN; a TN of register type is a virtual register

required to be allocated and a TN of immediate type is converted to a literal value of assembly format. Assigning a node n to use u type of functional units is denoted by $n^u, u \in \{M, I, B\}$ and assigning a TN t to be allocated in the register file namely f is denoted by $t^f, f \in \{D, A, AC, R, C\}$. If we define $TN(k)$ is the TNs of the node n_k, we will make a functional unit assignment of v nodes that may utilize as many local register files as possible by the strategies as follows.

- We prefer to utilize M-unit and I-unit as more as possible than B-unit due to the fact that more instruction-level-parallelism may be exploited between instructions of M-unit and I-unit by intra-cluster manner or inter-cluster manner. M-unit and I-unit also have more register resources, which are beneficial to optimize scheduling, comparing to B-unit.
- Since M-unit has direct accessibility to memory and inter-cluster communication, if both M-unit and I-unit are capable of executing an instruction, we have much favor to M-unit because it may cause fewer communication code to be inserted in the final phase of PALF register allocation.
- Instructions using the same local register file in the same cluster cannot be executed in parallel forever so that if two instructions have a data dependency, allocating the data in local register files will never be worse than in global register files. Also, instructions with less global register file usage should have less interference between scheduling and register allocation.

By the strategies stated above, we use the following steps to greedily assign the v nodes to the appropriate functional unit denoted as u, by the priority of $u \equiv M$, $u \equiv I$, $u \equiv M$, $u \equiv I$, ..., and finally $u \equiv B$:

1. Let Ψ be the set of nodes unassigned to any functional unit.
2. Select a maximal set of s nodes $S = \{n_{p_1}, n_{p_2}, \ldots, n_{p_s}\} \subseteq \Psi$ that could be assigned to use the same functional unit denoted as u, whereas $\forall n_{p_q}, 1 \leq q < s, \exists (t_i, t_j), 1 \leq i \leq r, 1 \leq j \leq r, t_i \in TN(p_q), t_j \in TN(p_{q+1})$ so that produces a precedence ordering to make these s nodes unable to be parallelized.
3. Assign all s nodes to u, denoted as $S = \{{n_{p_1}}^u, {n_{p_2}}^u, \ldots, {n_{p_s}}^u\}$ and remove S from Ψ.
4. Repeat steps 1–3 until $\Psi = \emptyset$.

4.3 Register File Assignment

After determining the functional unit type of all instructions, we are ready for assigning the register file used for each TN in G. The sequence of assignment includes: locating the TNs that must be allocated in global D-register files first to avoid unnecessary communication code caused by data sharing between different functional units, and then letting other TNs be allocated to the proper local registers associated to their functional unit assignments. While we assign all TNs to either of global register files or local register files, we may optionally try to use constant register files instead of global or local register file in assignments of the TNs that are capable of this replacement, to aggressively reduce the possible

register pressure of global and local register files if the compiler option is set to utilize the constant registers by software developers.

The assignment steps are detailed as follows.

1. Let Ω be the set of TNs that are not assigned to any register file.
2. $\forall(t_i, t_i), 1 \leq i \leq r$, whereas $t_i \in TN(l) \cap TN(m), 1 \leq l \leq v, 1 \leq m \leq v$ with either $\exists {n_l}^I {n_m}^M$ or $\exists {n_l}^M {n_m}^I$. Assign t_i to global D-register files, denoted as ${t_i}^D$.
3. Remove all ${t_i}^D, 1 \leq i \leq r$ from Ω.
4. Assign $t_i \in \Omega, 1 \leq i \leq r$ to the associated local register file of u, whereas $t_i \in TN(z), \forall {n_z}^u, 1 \leq z \leq v, u \in \{A, AC, R\}$.

4.4 Node Partitioning for Ping-pong Bank Assignment

To optimize the register allocation for ping-pong register files, we employ a partitioning procedure to determine which bank of ping-pong register file structures should be used for each TN assigned to D-register allocation. The partitioning for ping-pong bank assignment is developed to increase the opportunity of paralleliz-ing ping-pong bank access in the schedule; we will assign TNs whose associated instructions may interfere slightly with each other to different banks of D-register files. Let Δ be the set of w TNs, whereas $\forall t_i \in \Delta, 1 \leq i \leq r, \exists({t_i}^D, {t_i}^D)$ that ${t_i}^D \in TN(p) \cap TN(q)$ with assignments of ${n_p}^M, 1 \leq p \leq v$ and ${n_q}^I, 1 \leq q \leq v$. We partition Δ into two groups, X and Y, by the following methods, according to the threshold number λ of w:

- Build a new graph $G^* = (V^*, E^*)$, whereas $V^* \equiv \Delta$ and each edge $({t_i}^D, {t_j}^D)$ of E^* represents a node $n_z, 1 \leq z \leq v$ if $TN(z) \supseteq \{{t_i}^D, {t_j}^D\}, 1 \leq i \neq j \leq r$.
- If w is larger than λ, we apply stat-of-the-art multi-level k-way graph partitioning algorithms on G^* to get X and Y with the minimal number of $({t_i}^D, {t_j}^D)$ in G^*, whereas ${t_i}^D \in X, 1 \leq i \leq r, {t_j}^D \in Y, 1 \leq j \leq r$ and the size of the two groups is about equivalent.
- If w is smaller than or equal to λ, we partition Δ into two groups directly by their dependencies, moderating the communication code insertion between different ping-pong banks because the benefit of potential parallelism between too few instructions may not affordable to the degradation of any additional communication.

Currently the threshold λ is set as the total number of D-register banks.

4.5 Node Partitioning for Cluster Assignment

Depending on the compilation options set by software developers, compilers may generate code that utilizes two-clusters for performance optimizations or code that uses only one-cluster for low-power optimizations. In generating code that will be scheduled onto two clusters, we employ an iterative partitioning method based on cost-models to get a two-cut-sets of total graph G by the scheme as follows:

1. Given a CRTA-DDG $G = (V, R, E)$, partition G by native disjunction into k parts, $G = \bigcup {G^{\dagger}}_i, 1 \leq i \leq k$, whereas ${G^{\dagger}}_i \cap {G^{\dagger}}_j = \emptyset, \forall i \neq j, 1 \leq i \leq k, 1 \leq j \leq k$.
2. Group ${G^{\dagger}}_1, \ldots, {G^{\dagger}}_k$ into two separate sets, G° and $G^{\bullet}$, whereas G° and $G^{\bullet}$ are approximately balanced in overall schedule length by a cost-model consists of the factors in instruction number, critical path length, and dependency degree.
3. If any of G° and $G^{\bullet}$ is an empty set, or the difference of estimations in the cost-model by step 2 for G° and $G^{b}ullet$ is larger than a threshold, we try to find out some nodes that are feasible to be moved from the group with the longer predicted schedule length into the other group by evaluating another communication cost-model.
4. Annotate all edges in G across G° and $G^{\bullet}$ with the mark of pending inter-cluster communication code insertion.

4.6 Communication-Insertion/Postpass Register Allocation

If there are pending inter-cluster communication code generated in the last phase, we will insert the corresponding instructions. Moreover, for each edge $(t_i, t_j), 1 \leq i \leq r, 1 \leq j \leq r$, whereas $t_i \in TN(a), 1 \leq a \leq v$ and $t_j \in TN(b), 1 \leq b \leq v$, if the two nodes, ${n_a}^u$ and ${n_b}^{u^{\dagger}}$, whereas $u \neq u^{\dagger}$, we should check if the allocated registers with the edge (t_i, t_j) violate the constraints of PAC architectures. Finally, according to the register file assignments for all r TNs of v nodes in G, we apply register allocation based on graph-coloring heuristics for each register file to allocate each TN of register type to a physical register.

5 A PALF Register Allocation Example

In the following, we give a small example to illustrate how the PALF register allocation works. Fig. 6(a) shows the CRTA-DDG of an input program fragment: each rectangle labeled with its component-type association represents an operator, each circle represents an operand, or a TN in ORC, and each edge presents a data dependency between two operands; moreover, the color of a circle indicates the register-type association of the operand: green circles represent constant TNs, dark blue circles represent dedicated TNs, such as stack pointer TNs, and white circles represent register TNs whose register file is not assigned yet. Our goal is to determine an appropriate register file assignment — and thus a proper functional unit assignment and a feasible cluster assignment — in consideration of ping-pong register constraints and inter-cluster overheads.

Fig. 6(b)–6(g) illustrate the process of the first phase of the PALF register allocation scheme, the maximal localization. The main concept of the maximal localization is performing a greedy functional unit assignment that attempts to utilize as many local register files as possible and to distribute operations to M- and I-unit, roughly in equal amounts, as well in order to increase instruction level parallelism. It prefers that all nodes on a critical path, i.e., the path with

the maximum number of nodes, in the graph operate on the same functional unit so that their operands could be stored on local register files. Therefore, the maximal localization repeats the following process until all nodes in the graph are assigned with their own functional unit: it finds out the longest data-flow path, in which each node can be operated on M-unit, or I-unit, and its functional unit is not assigned yet, and enforces all nodes in the path to be operated on M-unit, or I-unit. In addition, the functional unit assignment alternates between M- and I-unit after each iteration to keep the amounts of nodes with M- and I-unit balanced. Fig. 6(b)–6(g) show the processing scenario. Rectangles with red border represent the nodes in the longest data-flow path and those with dark red border represent the nodes that have been assigned a functional unit.

Now we are ready for the register file assignment. Noticed that a register file assignment without considering ping-pong register constraints will harm in performance significantly as mentioned in Section 2. We must ascertain which TNs might create ping-pong conflicts and then attempt to eliminate or diminish the conflicts. The proposed approach is as follows. We first determine which edges connect two TNs that are operated on different functional units — the TNs must be allocated to a global register file so as to be accessed for the two operations — and then partition the TNs on those edges into two groups: one for ping-register assignments and the other for pong-register assignments. The rest of the unassigned register TNs are allocated to the corresponding local register files. Fig. 6(h) and 6(i) give the results of register file assignments: TNs with light blue, pink, and yellow color are allocated to ping, pong, and the corresponding local register files, respectively.

Next, we partition the nodes with M- and I-unit into two parts for the cluster assignment to take advantages of the two-cluster property of PAC DSP. Based on the approach described in Section 4.5, two critical paths, in which nodes are bordered with dark red and dark blue color, and their adjacent nodes are discovered. Fig. 6(j) presents the result of the cluster assignment: nodes with purple, sky blue, and orange color are assigned to cluster 1, cluster 2, and scalar unit, respectively. Fig. 6(k) gives the final result after inserting communication operations among cluster 1, cluster 2, and scalar unit. Fig. 6(l) is the same with Fig. 6(k) but each node is numbered for later explanations.

Fig. 7(a) and 7(b) illustrate the VLIW schedule of the running example for PAC DSP processor, respectively. Fig. 7(a) shows the schedule of a naive single-cluster approach that always assign an operation to M-unit if possible and always allocate the operand(s) of a M- or I-unit to the ping, or pong, register file if possible; Fig. 7(b) gives the schedule after performing the proposed PALF register allocation. The results show that the naive approach takes 19 cycles to complete the program with 1.1 of ICP (instructions per cycle), but the PALF approach only takes 11 cycles, in which the value of ICP is 3.4 for the first five cycles and 2.1 for overall. Noticed that it will take a 3-cycle delay for inter-cluster communications on PAC DSP.

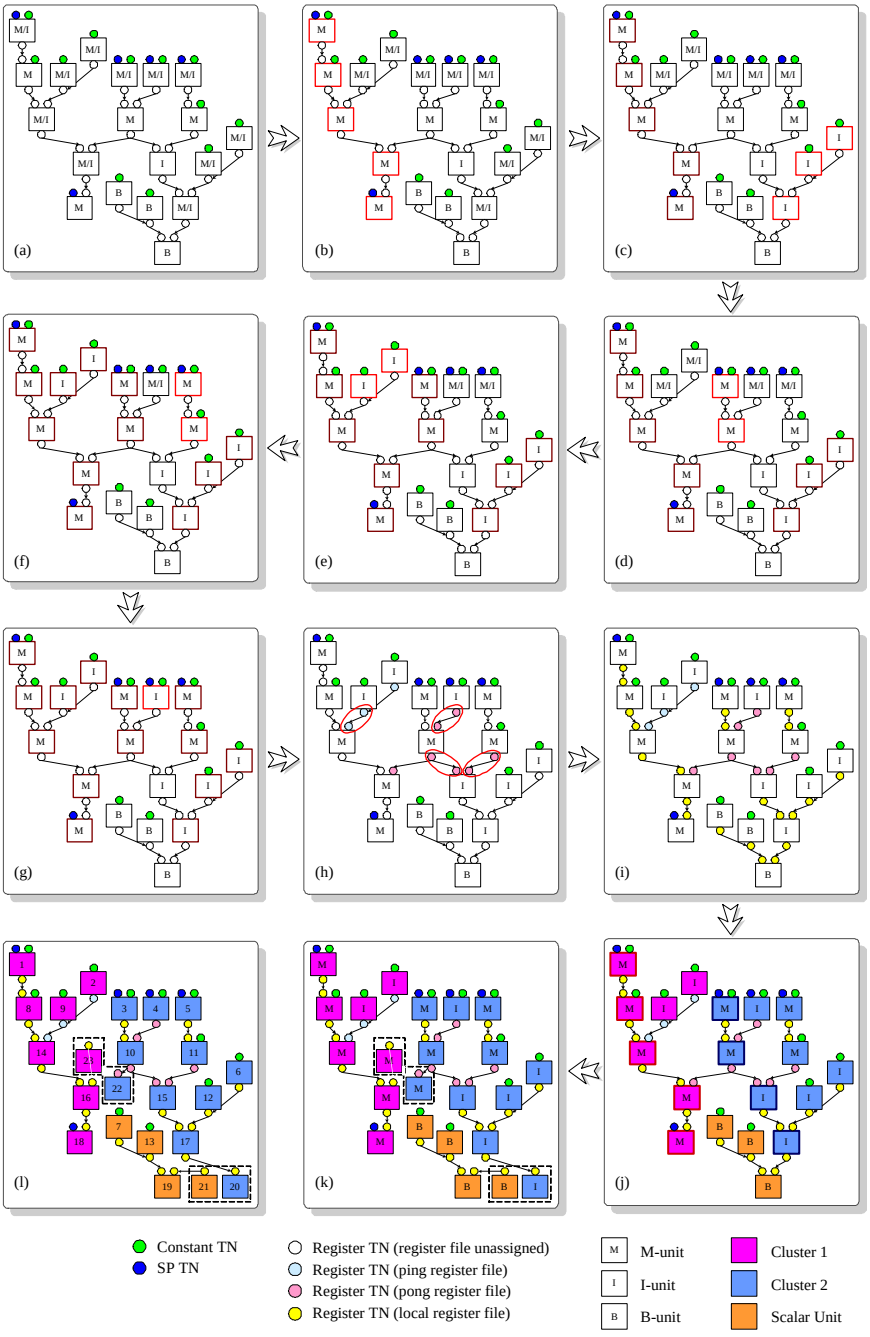

Fig. 6. A running example for PALF register allocation

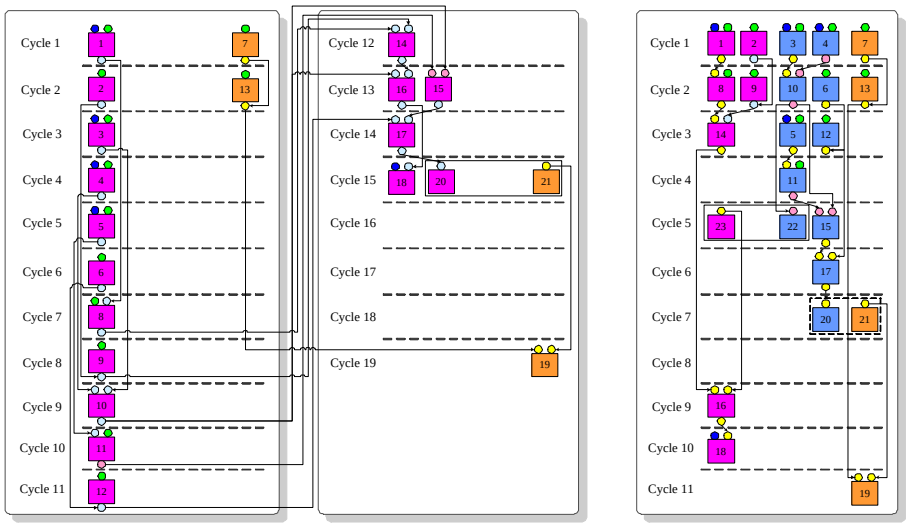

Fig. 7. The VLIW schedule of the running example for PAC DSP

6 Experiment and Discussion

We now examine the effectiveness of the proposed PALF register allocation with the DSPstone benchmark suite [15] on the PAC DSP processor. The PALF register allocation scheme is implemented on ORC infrastructure and the performance is evaluated with a cycle-accurate instruction set simulator.

Four register allocation schemes are evaluated, namely the traditional register allocation, the simulated annealing (SA) register allocation [11], the PALF register allocation using METIS graph partitioning library [7], implemented with Kernighan-Lin algorithm, for ping-pong bank assignment, and the PALF register allocation using the random scheme — vertices are assigned randomly to sets in a way that preserves balance — in CHACO graph partitioning library [5] for ping-pong bank assignment. Fig. 8 illustrates the normalized simulated execution time, using traditional register allocation as the baseline. It shows that our PALF approach has average 22.8% and 18.0% reduction in execution time to the traditional one when using METIS and CHACO graph partitioning library in the third phase of the PALF register allocation scheme respectively while the SA register allocation has average 32.5%, which is considered as a lower-bound since simulated annealing is nearly an exhausted approach. The performance result of the PALF scheme is close to that of the SA approach for most of the benchmarks. However, for some cases, like *complex_multiply*, *complex_update*, *dot_product*, *biquad_one_section*, and *real_update*, the performance of PALF is much farther from that of SA and even worse than that of the traditional. It is observed that the simulated execution time of those benchmarks are only hundreds of cycles (from 159 to 497 cycles) which weakens the capability of PALF since the programs are too small for optimizations in the PALF manner. Excluding the

mentioned benchmarks, the average reductions of PALF-metis and PALF-chaco-random become 36% and 31.3% which are closer to 43.5% of SA. Furthermore, the figure also elaborates that performance varies if different graph partitioning methods in the phase of the ping-pong bank assignment are applied. The result of PALF-metis is always better than that of PALF-chaco-random. The reason is that the Kernighan-Lin algorithm in PALF-metis attempts to partition a graph into two balanced parts with the minimal number of edge cuts but in PALF-chaco-random the graph is partitioned in a random way. By our definition, an edge represents a data-flow between a M-unit and an I-unit; if the two TNs spanned by the edge are allocated to ping and pong register bank respectively, it might require additional time to propagate the data because of the ping-pong register constraints.

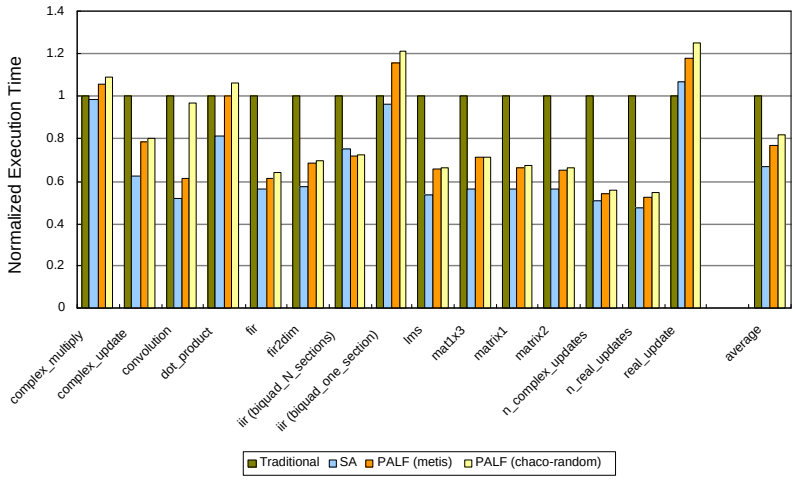

Fig. 8. Normalized execution time for various register allocation schemes

7 Conclusion

Embedded DSP processors are currently designed to exploit high instruction-level-parallelism with technological constraints in terms of cycle time, power dissipation, and die area. The techniques used in their designs commonly tend to include a clustered/partitioned architecture and low-power register file structures. In this work, we developed and implemented a novel heuristic approach for generating code for PAC VLIW DSP processors that incorporates highly-partitioned register files with the special ping-pong structure design. At the heart of this work is a proposed register file/bank assignment scheme which may integrate with the existing unified register allocation methodologies to provide a feasible solution. The experimental evaluation of several benchmark programs

indicates that our register allocation scheme for PAC VLIW DSP processors well utilizes all register files and delivers comparable results to a simulated-annealing approach which takes very long compilation time inestimable to get near-optimal solutions.

References

1. Aho, A. V., J. D. Ullman, and R. Sethi: Compilers Principles, Techniques, and Tools. Addison-Wesley, Reading, MA, 1986.
2. CEVA: CEVA-X1620 Datasheet. CEVA, 2004.
3. David Chang and Max Baron: Taiwan's Roadmap to Leadership in Design. Microprocessor Report, In-Stat/MDR, Dec. 2004. http://www.mdronline.com/mpr/archive/mpr_2004.html
4. A. Capitanio, N. Dutt, and A. Nicolau: Partitioned Register Files for VLIW's: A Preliminary Analysis of Tradeoffs. Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO-25), pages 292–300, Portland, OR, December 1–4 1992.
5. B. Hendrickson and R. Leland: The Chaco user's guide, version 2.0. Tech Report SAND95-2344, Sandia National Laboratories, Albuquerque, NM, October, 1994.
6. R. Ju, S. Chan, and C. Wu: Open Research Compiler for the Itanium Family. Tutorial at the 34th Annual Int'l Symposium on Microarchitecture, Dec. 2001.
7. George Karypis and Vipin Kumar: A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM J. Scientific Computing, 20(1): 359–392, 1999.
8. T.-J. Lin, C.-C. Lee, C.-W. Liu, and C.-W. Jen: A Novel Register Organization for VLIW Digital Signal Processors. Proceedings of 2005 IEEE International Symposium on VLSI Design, Automation, and Test, pages 335–338, 2005.
9. T.-J. Lin, C.-C. Chang. C.-C. Lee, and C.-W. Jen: An Efficient VLIW DSP Architecture for Baseband Processing. Proceedings of the 21th International Conference on Computer Design, 2003.
10. Tay-Jyi Lin, Chie-Min Chao, Chia-Hsien Liu, Pi-Chen Hsiao, Shin-Kai Chen, Li-Chun Lin, Chih-Wei Liu, Chein-Wei Jen: Computer architecture: A unified processor architecture for RISC & VLIW DSP. Proceedings of the 15th ACM Great Lakes symposium on VLSI, April 2005.
11. Yung-Chia Lin, Chung-Lin Tang, Chung-Ju Wu, Ming-Yu Hung, Yi-Ping You, Ya-Chiao Moo, Sheng-Yuan Chen, and Jenq Kuen Lee: Compiler Supports and Optimizations for PAC VLIW DSP Processors. Proceedings of the 18th International Workshop on Languages and Compilers for Parallel Computing, 2005.
12. R. A. Ravindran, R. M. Senger, E. D. Marsman, G. S. Dasika, M. R. Guthaus, S. A. Mahlke, and R. B. Brown: Increasing the number of effective registers in a low-power processor using a windowed register file. Proceedings of the 2003 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '03), 125–136, 2003.
13. S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens: Register organization for media processing. International Symposium on High Performance Computer Architecture (HPCA), pp.375-386, 2000.
14. Texas Instruments: TMS320C64x Technical Overview. Texas Instruments, Feb 2000.
15. V. Zivojnovic, J. Martinez, C. Schlager, and H. Meyr: DSPstone: A DSP-oriented benchmarking methodology. Proceedings of the International Conference on Signal Processing and Technology, 715–720, 1995.