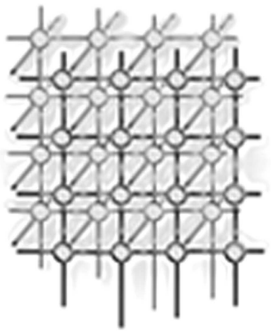
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*



# LC-GRFA: global register file assignment with local consciousness for VLIW DSP processors with irregular register files

Chia-Han Lu, Yung-Chia Lin, Yi-Ping You, and Jenq-Kuen Lee*†

*Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan*

**SUMMARY**

**Embedded processors developed within the past few years have employed novel hardware designs to reduce the ever-growing complexity, power dissipation, and die area. While using a distributed register file architecture with irregular accessing constraints is considered to have less read/write ports than using traditional unified register file structures, conventional compilation techniques can not produce the optimal performance from such new register file organizations. This paper presents a novel scheme for register allocation that includes global and local components on a VLIW DSP processor with distributed register files whose port access is highly restricted. In the scheme, a optimization phase performed prior to conventional global/local register allocation, named global/local register file assignment (RFA), is used to minimize various register file communication costs. For a register file structure where each cluster contains heterogeneous register files, the enchancement required to conventional register allocation scheme with cluster assignment is for it to cope with both inter- and intra-cluster communications. Due to the potential heavy influences of global RFA on local RFA, a heuristic algorithm is proposed for global RFA to make suitable decisions on communication for local RFA. Experiments were performed with a compiler developed based on the ORC (Open Research Compiler), with the results indicating that compilation based on proposed approach delivers significant performance improvements, comparable to the solution using only the PALF scheme that we have developed previously.**

KEY WORDS: register allocation; ping-pong register file; DSP; VLIW

*Correspondence to: Jenq-Kuen Lee, Department of Computer Science, National Tsing-Hua University, Hsinchu 30013, Taiwan.
†E-mail: jklee@cs.nthu.edu.tw
Contract/grant sponsor: XXXXXXXXXXXXXXXXXXXXXXXXX; contract/grant number: XXXXXXXXX

## 1. INTRODUCTION

Embedded processors developed in recent years have attempted to employ novel hardware designs to reduce the ever-growing complexity, power dissipation, and die area. While using a distributed register file architecture with irregular accessing constraints is considered to have less read/write ports than traditional unified register file structures, conventional compilation techniques can not produce the optimal performance from such new register file organizations. Our research work is focused on compiler schemes for a VLIW DSP with distributed register files, known as PAC (Parallel Architecture Core) architectures.

The appearance of multiple banks of register files, distributed register clusters, and ping-pong architectures on embedded VLIW DSPs (e.g., PAC architectures) presents a great challenge for compilers attempting to generate efficient codes for multimedia applications. Current research results on compiler optimizations for such problems in the literature have been limited to addressing issues related to cluster-based architectures. This includes work on partitioning register files to work with instruction scheduling [19], loop partitions for clustered register files [20], and cluster register files [17]. This complex optimization issue for embedded DSPs has been addressed previously [16], but only in the layer of copy propagation optimizations, and attempts have been made to deal with software pipelining issues with distributed register files [21] In this paper, we address the issues related to global register allocation.

This work relates to a compiler based on a PAC DSP [3, 7, 8] designed with distinctively banked register files where port access is highly restricted. The PAC DSP employs a heterogeneous design with a single scalar unit (for light-weight arithmetic, address calculation, and program flow control) plus two data-stream-processing clusters, each of which contains a load/store unit and a ALU/MAC unit with powerful SIMD capabilities; every unit in the clusters collocates three diverse types of register files, providing different accessing methods and constraints, and the scalar unit has its own accessible register file. The major specialization of the register file architectures in the PAC DSP is the incorporation of a so-called *ping-pong register file structure* [6]. This is divided into two banks, each of which can only be accessed mutually exclusively, with a semi-centralized register file among clusters and functional units within a cluster. This design decreases the power consumption due to there being fewer port connections, but its clustered nature makes register access across clusters problematic, and the switched access nature of the ping-pong register file prompts the investigation of further register assignment methods for increasing the degree of instruction-level parallelism.

This paper presents a novel scheme for register allocation comprising global and local register allocation on a VLIW DSP processor with distributed register files whose port access is highly restricted. In the scheme, a phase performed prior to conventional global/local register allocation (GRA/LRA), named global/local RFA (register file assignment), is introduced to minimize various register file communication costs. Our work is based on the Open Research Compiler (ORC) software framework, and in which register allocation comprises GRA and LRA. There are two kinds of temporary name (TN): local and global. LRA manages to allocate registers for local TNs, whose liveness is bounded within a single basic block, and

GRA does this for global TNs (GTNs), whose liveness will cross basic blocks. GRA proceeds before LRA in the back-end of the ORC. For featured register file structures where each cluster contains heterogeneous register files, the conventional register allocation scheme with the cluster assignment has to be enhanced to cope with both inter-cluster and intra-cluster communications. Due to the potential heavy influences of global RFA (GRFA) on local RFA (LRFA), a heuristic algorithm is proposed where GRFA makes suitable decisions on communication for LRFA. Experiments were performed with a compiler developed based on the ORC, with the results indicating that compilation based on proposed approach delivers significant performance improvements, comparable to the solution using only the ping-pong aware local favorable (PALF) scheme that we have developed previously [14, 15].

The remainder of this paper is organized as follows. In Section 2, we introduce the processor architecture and register file structure of the PAC VLIW DSP. The proposed register allocation scheme involving RFA is covered in Section 3. Section 4 compares register allocation schemes, and Section 5 presents related work on cluster assignment and our previous work on the processor. Finally, conclusions are drawn in Section 6.

## 2. PAC DSP Architecture

The PAC DSP features a clustered VLIW architecture that boosts scalability, and a large number of registers arranged as innovative heterogeneous and distinct partitioned register file structures. Unlike the symmetric architectures of most DSPs available nowadays, the PAC DSP is constructed as a heterogeneous five-way-issue VLIW architecture, comprising two integer ALUs (I-unit), two memory load/store units (M-unit), and the program sequence control unit/scalar unit (B-unit) that is mainly in charge of control-flow instructions such as branch and jump. Each unit has its own executable subset of the instruction set, and each executable instruction is subject to its own register accessibility constraints. The M- and I-units are organized in pairs, with each pair containing one M-unit and one I-unit in a cluster with associated register files. Each cluster is logically appropriate for processing one data stream, and the current design of the PAC DSP consists of two clusters to support a maximum workload capacity of two concurrent data streams. However, the scalability of the cluster design in the PAC DSP could allow the processor to use more clusters to increase the data processing workload. The B-unit consists of two subcomponents – the program sequence control unit and the scalar unit – due to the hierarchical decoder design for variable-length instruction encoding in the PAC DSP. The program sequence control unit is primarily responsible for 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 presented in Fig. 1.

Registers in the 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 as to minimize chip area and power consumption. Local register files are attached to all units in the processor, including the R-register file, AC-register file, and A-register file, which are only accessible by the B-, I-, and M-units, respectively. Global register file named the D-register file is designed to be shared by the pair of M- and I-units in each cluster. The D register

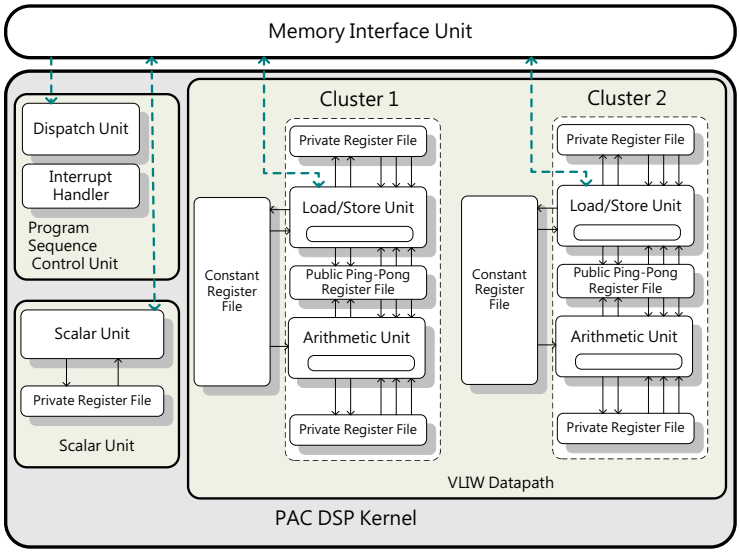



Figure 1. PAC DSP architecture.

file is further partitioned into two banks to utilize instructional port switching technology in order to further reduce wire connections between the M- and I-units. This technology is the *ping-pong register file structure*, which decreases the register bank port connections that limit the accessibility of the two banks; in each cycle, the two functional units must access different banks. Each instruction bundle encodes the information of which bank is to be accessed by each functional unit in the cycle, so that the hardware can perform port switching between D-register file banks and functional units to facilitate data sharing within a cluster. Overlapping two data-stream operations in a cluster minimizes occasions when M- and I-units access the same data at the same time, and hence the access constraints of the ping-pong register file structure should have little impact on the performance. The ping-pong register file structure design is to consume less power due to the reduced number of read/write ports [11] while retaining the data communication capability. Each cluster also contains an additional constant register file that 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. Local-Conscious Global Register Allocation Scheme

This section describes our proposed local-conscious GRFA (LC-GRFA), including the adaptation of our previously proposed PALF-LRFA, which is referred to the LRFA used in the PALF scheme.

### 3.1. Motivation and Register File Assignment

Because the PAC DSP has a clustered VLIW structure with irregular distributed register files, conventional cluster assignment (CA) does not produce efficient communications in the DSP. These communications can be divided into intra-cluster and inter-cluster communications. Intra-cluster communication, which occurs within a single cluster, uses the public register file as the communication channel. For functional unit $FU_a$, if there is an operand in the same cluster that it cannot access, (i.e., belonging to the private register file of another functional unit, called $FU_b$), intra-cluster communication is required, and hence data must be copied from the private register file of $FU_b$ into the public register file to make it accessible to $FU_a$. Furthermore, inter-cluster communication will also be required if an operand needed by $FU_a$ is on the other cluster. Moreover, one cluster must issue a *send* instruction and the other one must issue a *receive* instruction at the same time (in the same bundle) to complete the inter-cluster communication.

Fig. 2 illustrates the complicated communication in the PAC DSP. In Fig. 2a the original instruction is an addition and is unbundlized, while in Fig. 2b the instruction has its TN register files assigned. Then, Fig. 2c has a bundle where the instruction is bound to **I2**. However, because **I2** cannot access *1A*, inter- and intra-cluster communication is required. Thus, in Fig. 2d, a bundle that contains *send/receive* instructions is inserted for inter-cluster communication, which copies from *1A* to *2A*, while a *nop* bundle has to follow to meet the latency requirement of these instructions. Although there is an inter-cluster copy, intra-cluster communication is necessary because *2A* is still out of reach of **I2**. This leads to a bundle including a copy, as shown in Fig. 2e.

Complicated communication scheme in the PAC DSP makes it desirable to use a new phase, RFA, to handle communications. RFA includes three major subphases: CA, register file class assignment, and copy insertion. The first two subphases handle inter- and intra-cluster communications and assign suitable register files to TNs, while the third subphase inserts and unfolds copy instructions to establish the communications derived in the other subphases. The PAC DSP compiler, derived from the ORC, also divides register allocation into GRA and LRA, which are register allocations for GTNs and TNs, respectively. Due to various differences between GRA and LRA, the mentioned RFA has to be extended into GRFA and LRFA, as shown in Fig. 3.

### 3.2. Ping-pong Aware Local Favorable Register Allocation Scheme

This section reviews our previously proposed register allocation algorithm that, given a dependency DAG (directed acyclic graph) [1] describing the compilation regions, heuristically determines the appropriate register file/bank assignment and employs state-of-the-art graph-coloring register allocation for each assigned register file/bank in PAC architectures.

The proposed overall register allocation algorithm is shown in Fig. 4. Our approach requires building an extended data dependence DAG, 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 the extended data dependence DAG represent instructions of the input code block, with the component-type association

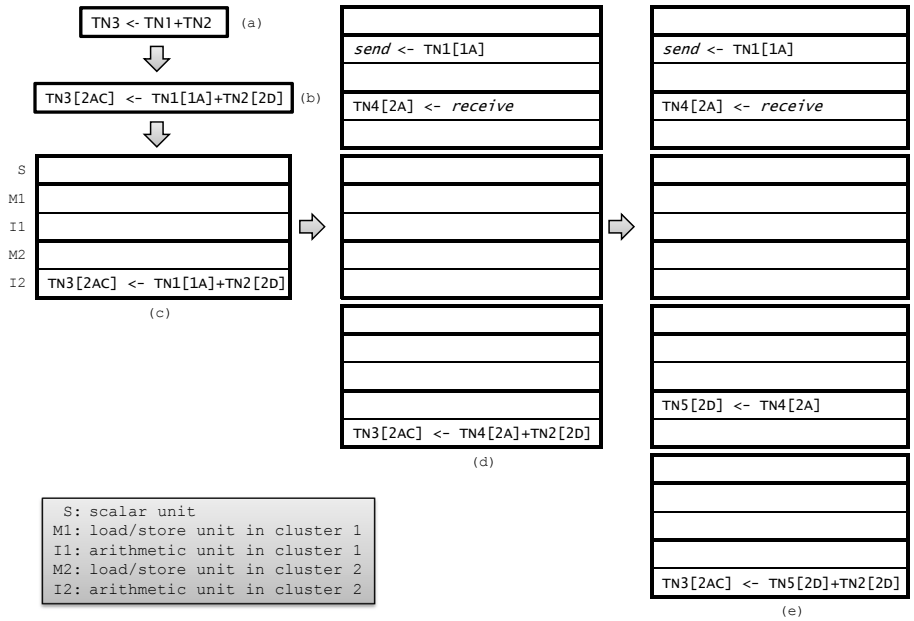



Figure 2. Example of inter-/intra-cluster communication in the PAC DSP: (a) original instruction, (b) each TN has its register file assigned, (c) bundlized, (d) after inter-cluster communication insertion, and (e) after intra-cluster communication insertion. In the figure, $A$ is the private register file of the M-unit, $AC$ is the private register file of the I-unit, and $D$ is the public register file, and $A$, $AC$, and $D$ are together defined as the *register file class* of TN, which are prefixed with a number representing the cluster. Thus, *1D* is the public register file in cluster 1, and *2AC* is the private register file of the I-unit in cluster 2.

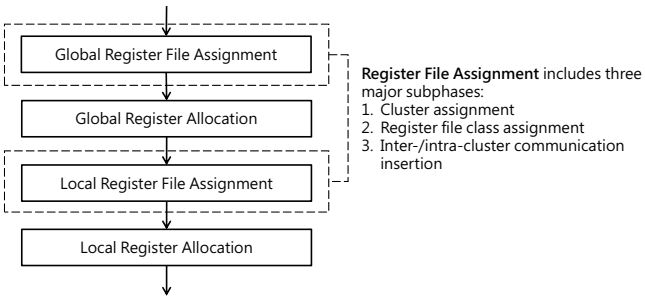



Figure 3. Register allocation scheme involving register file assignment with three major subphases.

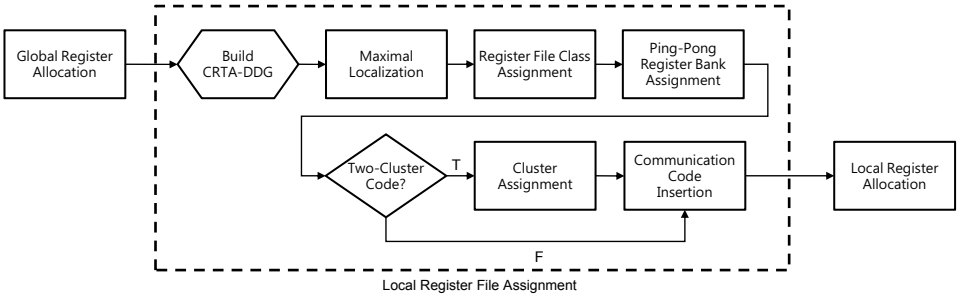



Figure 4. Flowchart of PALF scheme.

(that indicates which functional unit should preferentially 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. The main PALF register allocation scheme could be organized into the following five phases:

1. Perform the preferable functional unit assignment to the default execution type of each instruction using *maximal localization* analysis.
2. Assign operands/results (required to be allocated to physical registers) of each node in the extended data dependence DAG to the desirable register files.
3. Partition the operands/results assigned to the global ping-pong register files to the preferred register banks according to the strategy of optimizing ping-pong parallelism.
4. Optionally partition the nodes in the extended data dependence DAG into two clusters properly if we would like to compile for two-clusters.
5. Insert nodes of the required communication code to avoid invalid register file/bank assignment and cluster partitioning, followed by the physical register allocation for each register file.

### 3.3. Extending Local Register File Assignment for Global Register Allocation

Due to the variety of distributed register files involved and irregular access constraints in the design of PAC DSP architectures, we developed the PALF scheme to include an appropriate LRFA phase in the compiler developed for the PAC DSP before the actual LRA, allowing state-of-the-art graph-coloring-based methods for allocating homogeneous registers in the same register file. This design simplifies the optimization of register allocation for the irregular and distributed register files used in the PAC DSP, but still achieves the expected performance.

Although the proposed PALF-LRFA performs excellently with the local optimizations and instruction scheduling, GRFA cannot be directly derived from the PALF-LRFA since the original design decisions in the heuristics is based on the data dependence DAG. Therefore,

we add a new phase of GRFA, named LC-GRFA, which refers to the local preference of the PALF-LRFA for each basic block and determines the appropriate RFA for GTNs across multiple blocks. Our proposed LC-GRFA extends the PALF-LRFA to obtain RFA of GTNs that is advantageous for both LRFA and local instruction scheduling. We now introduce two LC-GRFA methods: one-pass and two-pass.

The one-pass GRFA is proposed in Algorithm 1, which includes the following steps:

**Algorithm 1** One-pass LC-GRFA

**Require:** $\mathsf{BB\_List_{all}}$ and $\mathsf{GTN\_List_{all}}$
**Ensure:** Register file assignment of all GTNs
  {*Step 1: Basic block prioritization*}
  $\mathsf{BB\_List_{prio}} \leftarrow$ Prioritize the basic blocks in $\mathsf{BB\_List_{all}}$;
  {*Step 2: Register file assignment*}
  **while** ($\mathsf{BB\_List_{prio}}$ is not empty) or ($\mathsf{GTN\_List_{all}}$ is not empty) **do**
    $\mathsf{BB_{prio}} \leftarrow$ Pop the basic block from $\mathsf{BB\_List_{all}}$ with the highest priority;
    $\mathsf{GTN\_List_{fixed}} \leftarrow$ All the global temporary names whose register files have been fixed in $\mathsf{GTN\_List_{all}}$;
    $\mathsf{Reg\_File\_Assign_{prio}} \leftarrow$ *Local_Reg_File_Assign*($\mathsf{BB_{prio}}$, $\mathsf{GTN\_List_{fixed}}$);
    **for all** $\mathsf{GTN_{prio}} \in$ *GTN_List*($\mathsf{Reg\_File\_Assign_{prio}}$) **do**
      Fix register file of $\mathsf{GTN_{prio}}$;
    **end for**
  **end while**
  {*Step 3: Communication insertion*}
  **for all** $\mathsf{BB} \in \mathsf{BB\_List_{all}}$ **do**
    *Insert_Comm*(BB);
  **end for**

1. *Basic block prioritization.* The prioritization will prioritize basic blocks in the programming unit based on static or dynamic score. The static score is calculated based on factors such as the loop depth, scheduling length, while the dynamic score is based on profiling. Basic blocks with higher scores have a greater impact on overall performance, so they should be given higher priority to be processed earlier.
2. *RFA.* This step employs LRFA virtually on basic blocks in order of priority until all basic blocks are traversed or all GTNs have register files. For each iteration, if there are previous assignments to GTN register files, the iteration uses them as a precondition for the LRFA to be employed. After the LRFA, the step preserves GTN register files given by LRFA and clears those of TNs. Because GRA will produce extra TNs, it is preferable to leave all TN register files to be assigned in LRFA after GRA.
3. *Communication insertion.* The third step involves simply inserting intra- or inter-cluster communication codes so as to make all operations legal.

In addition to the one-pass method, we propose the two-pass method that employs separate two passes of register file class assignment and CA. Since either passes refer LRFA to make

their assignment, the method is considered to be advantageous for local optimizations in some cases. This method is displayed in Algorithm 2, which mainly includes the following passes:

**Algorithm 2** Two-pass LC-GRFA

**Require:** $\mathsf{BB\_List_{all}}$ and $\mathsf{GTN\_List_{all}}$
**Ensure:** Register file assignment of all GTNs
  {*Pass 1: Register file class assignment*}
  {*1.1: Local register file assignment collection of basic blocks*}
  Reg_File_Class_Assign_List ← *Empty*;
  **for all** BB ∈ $\mathsf{BB\_List_{all}}$ **do**
    Reg_File_Class_Assign_List [BB]
    ← *Reg_File_Class_Assign*(*Local_Reg_File_Assign*(BB));
  **end for**
  {*1.2: Register file class assignment*}
  **for all** GTN ∈ $\mathsf{GTN\_List_{all}}$ **do**
    Reg_File_Class ← *Consistent_Reg_File_Class*(GTN, Reg_File_Class_Assign_List);
    **if** (Reg_File_Class is *Nothing*) **then**
      *Reg_File_Class*(GTN) ← Reg_File_Class;
    **else**
      *Reg_File_Class*(GTN) ← *Public Register File Class*;
    **end if**
    Fix register file class of GTN;
  **end for**
  {*Pass 2: Cluster assignment*}
  {*2.1: Basic block prioritization*}
  $\mathsf{BB\_List_{prio}}$ ← Prioritize the basic blocks in $\mathsf{BB\_List_{all}}$;
  {*2.2: Cluster assignment*}
  **while** ($\mathsf{BB\_List_{prio}}$ is not empty) or ($\mathsf{GTN\_List_{all}}$ is not empty) **do**
    $\mathsf{BB_{prio}}$ ← Pop the basic block from $\mathsf{BB\_List_{all}}$ with the highest priority;
    $\mathsf{GTN\_List_{fixed}}$ ← All the global temporary names whose clusters or cluster file classes have been fixed in $\mathsf{GTN\_List_{all}}$;
    $\mathsf{Cluster\_Assign_{prio}}$ ← *Cluster_Assign*(*Local_Reg_File_Assign*($\mathsf{BB_{prio}}$, $\mathsf{GTN\_List_{fixed}}$));
    **for all** $\mathsf{GTN_{prio}}$ ∈ *GTN_List*($\mathsf{Cluster\_Assign_{prio}}$) **do**
      Fix cluster of $\mathsf{GTN_{prio}}$;
    **end for**
  **end while**
  {*Communication insertion*}
  **for all** BB ∈ $\mathsf{BB\_List_{all}}$ **do**
    *Insert_Comm*(BB);
  **end for**

1. *Register file class assignment.* In the first pass, the method manages to assign register file classes to all GTNs, such as $A$, $AC$, and $D$. After colleting LRFA of each basic blocks,

it uses the information to assign the GTN register file class. A GTN is 'consistent' in register file class assignment if LRFA of each basic block assigns the GTN to the same register file class. If this occurs, it indicates that LRFA may agree to the consistent register file class, and hence the method assigns the GTN to the class. Otherwise, it assigns the GTN to the public register file class since the disagreement suggests that intra-communication may be required.

2. *Cluster assignment.* After all GTN register file classes are assigned, CA is performed in the second pass, which operates like the one-pass method.

### 3.4. Examples of Local-Conscious Global Register File Assignment

This section provides simple examples to illustrate the current version of the proposed LC-GRFA methods. Fig. 5 shows an example of the one-pass method with 10 GTNs in the programming unit, which are listed in the leftmost column. And each assignment of GTN comprises two parts: the cluster and the register file class. For example, an assignment of a GTN to $D$ in cluster 2 is represented by *2D*. The one-pass method is based on the basic block priority. Referring to Fig. 5, in Step 1 the method chooses BB5 (which has the highest priority), which references four GTNs: GTN4, GTN5, GTN6, and GTN12. The method performs LRFA and assigns the GTNs to *1D* and *2D*. BB6 chosen in Step 2 references GTN5, GTN7, GTN8, GTN9, and GTN11. Because GTN5 has been assigned in Step 1, the GTN cannot be modified and used as a precondition for subsequent LRFA. So, LRFA is performed in Step 2 using the assignment of GTN5 as a precondition, with GTN7, GTN8, GTN9, and GTN11 being assigned. This process continues in Steps 3 and 4. At the end of Step 4 there are no more unassigned GTNs, and hence the method terminates.

Fig. 6 presents an example of the two-pass method. First of Pass 1 involves performing LRFA on each basic block independently and collecting the assignment of the register file class of each basic block as a reference. For example, BB2 references four GTNs: GTN1, GTN2, GTN3, and GTN6. LRFA produces an assignment of $A$ for GTN1 and GTN2, $AC$ for GTN3, and $D$ for GTN6. Pass 1 also makes a compromise among these assignments. For example, based on our method, Pass 1 assigns GTN1 to $A$, GTN6 to $D$, and GTN10 to $AC$. This is followed by Pass 2 performing CA based on Pass 1, which assigns GTN1 to *1A*, GTN6 to *2D*, and GTN10 to *1AC*.

## 4. Experiments and Discussion

This section describes our preliminary experiments on the proposed LC-GRFA methods. Fig. 7 compares various prioritization methods, while Fig. 8 compares various register allocation schemes using the DSPstone benchmark suite [13] in the PAC DSP, implemented on our developed compiler based on ORC infrastructures and evaluated with a cycle-accurate instruction set simulator of the PAC DSP.

Our method is based on prioritization of basic blocks, which influences the performance. Fig. 7 compares among three prioritization methods: block length, random, and frequency. The results for the random method are the averages of 10 runs of the benchmarks. The frequency

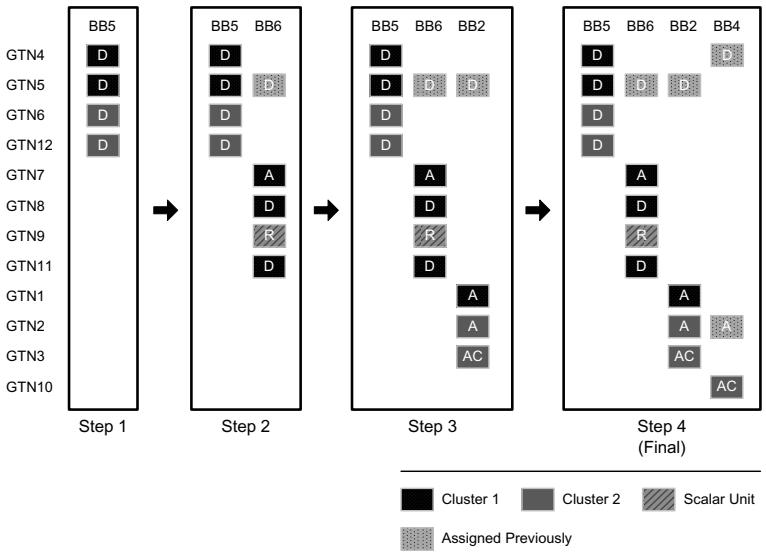



Figure 5. Example of one-pass LC-GRFA. In the figure, each column is headed by the basic blocks involved in each step.

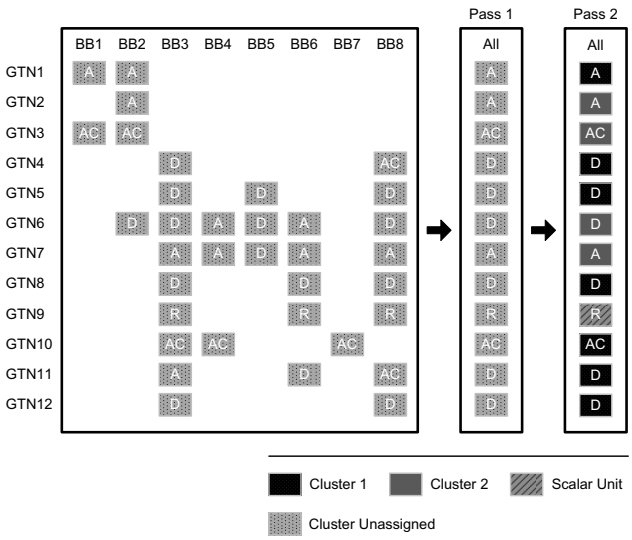



Figure 6. Example of two-pass LC-GRFA.

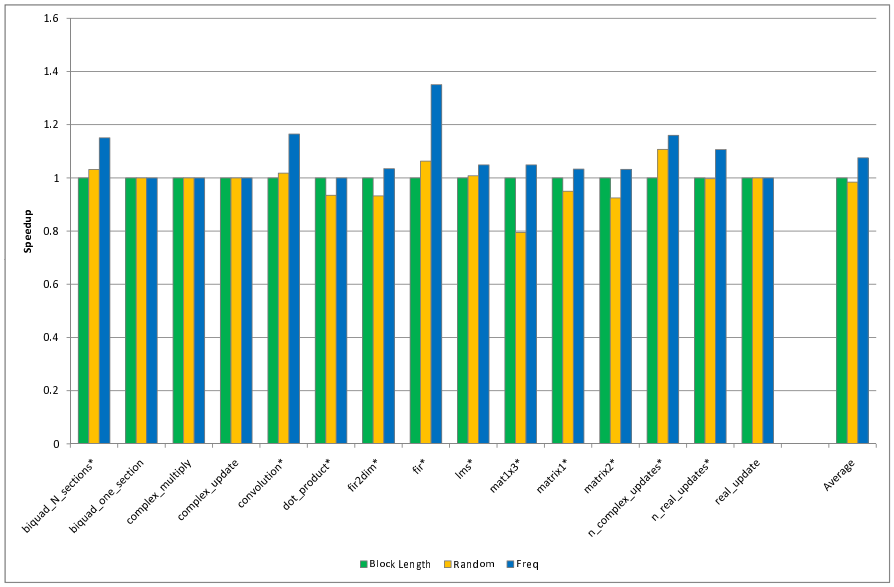



Figure 7. Comparison of various prioritization methods.

method refers to using the frequency utility provided by the ORC. Related to block-length method, the random method exhibits impaired performance whereas the frequency method improves performance by 7% on average.

The following three register allocation schemes with different RFA methods were compared: (1) PALF-LRFA only, (2) PALF-LRFA plus naïve GRFA, (3) PALF-LRFA plus the proposed one-pass LC-GFRA method, and (4) PALF-LRFA plus the proposed two-pass LC-GFRA method. The comparison indicates that the last two schemes provide 45% speedup on average relative to the PALF-LRFA and 30% speedup on average relative to the PALF-LRFA plus naïve GRFA.

Most of the programs in DSPstone benchmark suite have fine speedup using the scheme, except *n_complex_updates*. *n_complex_updates* reveals another issue in developing compilers for the PAC DSP architecture, which features an irregular distributed register file structure. For *n_complex_updates*, some register files are highly utilized in the proposed two-pass GFRA and lead to considerable spilling in the LRA, which degrades the performance for the benchmark. This could be solved using register pressure control in every register file in RFA to provide appropriate apportionment between LRA and GRA. Since the number of registers in register files on the PAC DSP is limited, the control relies on precise modeling of potential register usage scenarios.

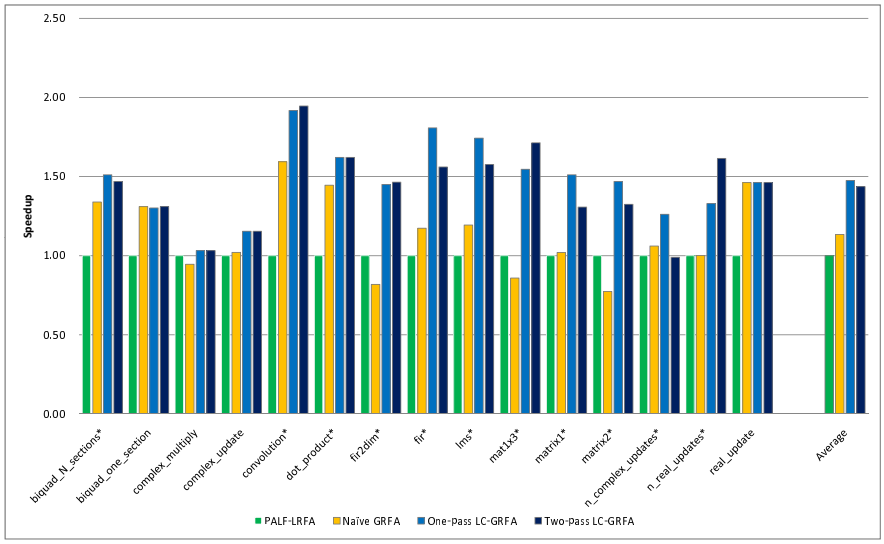



Figure 8. Comparison of register allocation schemes.

## 5. Related Work

Hiser, Carr, and Sweany have described global register partitioning for CA in [17], where a register component graph (RCG) is built on data dependence DAGs and an optimal schedule is used to model the relationship between registers. Every node in the RCG represents register operands, and the weight of an edge represents the 'affinity' the connected nodes: positive weight indicates that the related nodes should be put in the same cluster, and vice versa. They used a greedy heuristic algorithm to divide these nodes into two clusters. The experimental results show that their algorithm exhibits a performance degradation of only about 10% when compared to an unrealizable monolithic-register-bank architecture with the same level of ILP. However, their architecture is simpler than the PAC DSP which, which has is subject to the ping-pong constraint, and hence it is difficult to implement the algorithm in the PAC DSP

Terechko, Thénaff, and Corporaal provided another method for CA of global values [18]. Local values have short-lived ranges, while global values can be alive throughout the whole programming unit. Those authors provided several methods for performing the task, one of which is involved using the 'affinity' as mentioned above. The affinity between two global values indicates the benefit of assigning them to the same cluster based on the data flow graph. Equations were used to represent the affinity between two global values, rather than the RCG mentioned in [17] They also provided detailed experimental comparisons.

Besides PALF, we have previously proposed a simultaneous instruction scheduling/RFA method based on simulated annealing [9], which iteratively optimizes the entire RFA state of

a single basic block, and uses the instruction scheduler itself as the evaluation function. This local approach was borrowed from [19], and can also be globally extended in the same manner as PALF as presented in this paper.

In addition to work on the register allocation scheme, the copy propagation in the original ORC could also be modified [16]. Due to the irregular distributed register file structure in the PAC DSP, conventional copy propagation might degrade the performance. In the present study, a communication cost model was derived for copy propagation, which was based on the cluster distance, register port pressure, and movement type of register sets. The model was used to guide data flow analysis for better performance on the PAC DSP architecture. This scheme is effective at preventing the performance anomaly.

## 6. Conclusion

This paper proposes a new register allocation scheme involving a separated phase named RFA which comprises LRFA and GRFA. Due to the irregular distributed register file structures on PAC DSP architectures, where the conventional register allocation is not sufficient, RFA could form the core procedure of the scheme to boost performance on the architecture. The preliminary experimental results presented here show that the proposed scheme can efficiently utilize the distributed register file architectures and deliver good performance. Our future work will include a complete exploration of the various issues involved in our proposed scheme.

## 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. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.

11. 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.
12. Texas Instruments: TMS320C64x Technical Overview. Texas Instruments, Feb 2000.
13. 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.
14. Yung-Chia Lin, Yi-Ping You, and Jenq-Kuen Lee: Register Allocation for VLIW DSP Processors with Irregular Register Files. Proceedings of the 12th Workshop on Compilers for Parallel Computers (CPC 2006), Jan 9–11 2006.
15. Yung-Chia Lin, Yi-Ping You, and Jenq Kuen Lee: PALF: Compiler Supports for Irregular Register Files in Clustered VLIW Processors. Concurrency and Computation: Practice and Experience, 2007:19:1-16, Wiley, 2007.
16. Chung-Ju Wu, Sheng-Yuan Chen, and Jenq-Kuen Lee: Copy Propagation Optimizations for VLIW DSP Processors with Distributed Register Files Proceedings of the 19th International Workshop on Languages and Compilers for Parallel Computing, Nov. 2–4 2006.
17. Jason Hiser, Steve Carr, and Philip Sweany: Global Register Partitioning. Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques, 2000.
18. Andrei Terechko, Erwan Le Thénaff, and Henk Corporaal: Cluster Assignment of Global Values for Clustered VLIW Processors. Proceedings of the 2003 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '03), 32–40, 2003.
19. Rainer Leupers: Instruction scheduling for clustered VLIW DSPs. Proceedings of the 9th International Conference on Parallel Architecture and Compilation Techniques (PACT 2000), 291–300, Oct. 2000.
20. Yi Qian, Steve Carr, and Philip Sweany: Optimizing Loop Performance for Clustered VLIW Architectures. Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques (PACT 2002), Charlottesville, Virginia, Sep. 22–25, 2002.
21. Chung-Kai Chen, Ling-Hua Tseng, Shih-Chang Chen, Young-Jia Lin, Yi-Ping You, Chia-Han Lu, and Jenq-Kuen Lee: Enabling Compiler Flow for Embedded VLIW DSP Processors with Distributed Register Files. ACM SIGPLAN/SIGBED 2007 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'07), San Diego, Jun. 2007.