Integrating Compiler and System Toolkit Flow for Embedded VLIW DSP Processors

Chi Wu   Kun-Yuan Hsieh   Yung-Chia Lin   Chung-Ju Wu   Wen-Li Shih

S. C. Chen   Chung-Kai Chen   Chien-Ching Huang   Yi-Ping You   Jenq Kuen Lee

Department of Computer Science

National Tsing-Hua University

Hsinchu 300, Taiwan

Email: {wuchi, kyhsieh, yclin, jasonwu, wlshih, scchen, ckchen, cchuang, ypyou}@pllab.cs.nthu.edu.tw

Email: {jklee}@cs.nthu.edu.tw

Abstract

To support high-performance and low-power for multimedia applications and for hand-held devices, embedded VLIW DSP processors are of research focus. With the tight resource constraints, distributed register files, variable-length encodings for instructions, and special data paths are frequently adopted. This creates challenges to deploy software toolkits for new embedded DSP processors. This article presents our methods and experiences to develop software and toolkit flows for PAC (Parallel Architecture Core) VLIW DSP processors. Our toolkits include compilers, assemblers, debugger, and DSP micro-kernels. We first retarget Open Research Compiler (ORC) and toolkit chains for PAC VLIW DSP processor and address the issues to support distributed register files and ping-pong data paths for embedded VLIW DSP processors. Second, the linker and assembler are able to support variable length encoding schemes for DSP instructions. In addition, the debugger and DSP micro-kernel were designed to handle dual-core environments. The footprint of micro-kernel is also around 10K to address the code-size issues for embedded devices. We also present the experimental result in the compiler framework by incorporating software pipeline (SWP) policies for distributed register files in PAC architecture. Results indicated that our compiler framework gains performance improvement around 2.5 times against the code generated without our proposed optimizations.

1. Introduction

With the fast growth of consumer electronic products, the demand on digital signal processings related to the multimedia, such as image and video processing, is also in-creased significantly. To meet the performance challenges, high-end embedded processor and DSP processors are moving towards exploiting intensively instruction level parallelism (ILP). In addition, the tight resources in embedded systems also have DSP processors to adopt architecture features such as distributed register files to reduce the amount of read and write ports in registers to reduce power consumptions [1], special data path to exploit the characteristics of DSP applications, variable length encoding schemes for instructions to reduce code size, the sub-word instructions for video applications, etc. The appearance of these new features create challenges to deploy software toolkits for new embedded DSP processors.

This article presents our methods and experiences to develop software and toolkit flows for PAC (Parallel Architecture Core) VLIW DSP processors. Parallel Architecture Core (PAC) is a five-way VLIW DSP processors with distributed register cluster files and multi-bank register architectures (known as ping-pong architectures) [2] [3] [4]. Our toolkits include compilers, assemblers, debugger, and DSP micro-kernels. We first retarget Open Research Compiler (ORC) [5] and toolkit chains for PAC VLIW DSP processor and address the issues to support distributed register files and ping-pong data paths for embedded VLIW DSP processors. We also deploy software pipelining techniques with the considerations of distributed register file architectures. The linker and assembler of our toolkits are able to support variable length encoding schemes for DSP instructions. In addition, the debuggers were designed to handle dual-core environments. The debugger is also integrated with Eclipse IDE. The footprint of micro-kernel is also around 10K to address the code-size issues for embedded devices. We also present the experimental result in the compiler framework by incorporating software pipeline policies for distributed register files in PAC architecture. Results indicated that

our compiler framework gains performance improvement around 2.5 times against the code generated without our proposed advanced optimizations. In-depth technical experiences in deploying system toolkits for new embedded VLIW DSP processors are reported.

The remainder of this paper is organized as follows. Section 2 introduces the PAC VLIW DSP architecture and presents our software flow for this embedded DSP processor. Next, Section 3 presents compiler overview and our experiences in deploying software pipeline schemes for partitioned register files of PAC architectures. Section 4 briefly describes our experiences in the development of toolkit chains. Section 5 introduces an operating system called pCore which is designed to provide a minimal but sufficient OS services with a small kernel under the dual-core/multi-core environment on PAC architecture. The experimental results of our evaluation are provided in Section 6. Finally, Section 7 concludes this paper.

2. PAC architecture and Software Flow

The novel 32 bits embedded PAC DSP processor is designed for high-performance and low power. We briefly describe PAC's architecture in this section. In addition, we also present the software flow in our design for PAC platforms.

2.1. PAC architecture

PAC VLIW DSP processor is being developed by SOC Technology Center at Industrial Technology Research Institute with joint efforts from academic research teams [7] [8]. This high-performance processor with SIMD ISA is a five-way issue VLIW DSP processor which is constructed by two clusters and one scalar unit (B-unit). Each cluster contains one arithmetic unit (I-unit) and one load/store unit (M-unit) with associated register files as shown in Figure 1. Such architecture design is logically appropriate for data stream processing.
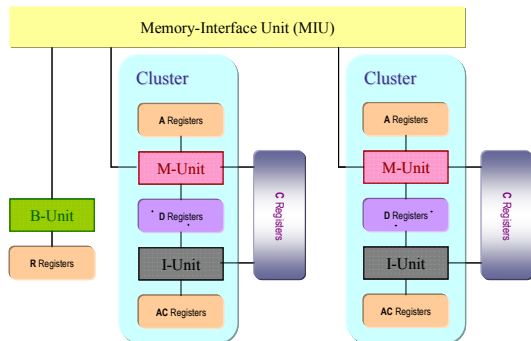

Figure 1. PAC VLIW DSP architecture

Only B and M-unit can do memory access through memory interface unit. A pair of M and I units form a cluster with their register files, and B-unit has its own local register file. There are several distinct register files in the architecture. They are five local register files, two global register files, and two constant register files. All three types of register files in the PAC DSP architecture have different access capabilities. Each local register file can only be accessed by its dedicated unit and each global register file is shared by a pair of M and I units but the access is limited by ping-pong switch constraints. Furthermore, constant register file in one cluster is also shared by a pair of M and I-units. However, it only be used as one read-only operand source for each unit and its value only setup by M-unit.

The ping-pong bank switches are the major features used by the PAC DSP architecture to reduce the number of ports but remain data-sharing capability for the global register files. Each global register file is divided into two banks, and each bank can only be accessed by one unit with one state of the ping-pong bank switch in a single cycle, ping state or pong state. The state of the ping-pong bank switches can be changed in each cycle. The data-transfer between M-unit and I-unit in same cluster is performed by ping-pong bank switch, shown as Figure 2.
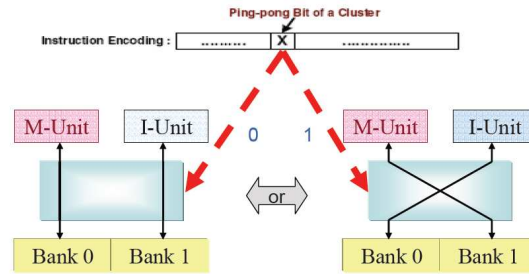

Figure 2. Behavior of global register files

The register file communication used by the PAC DSP architecture also features the reduction of port connections. Setting ping-pong bank switch on the global register file can make an intra-cluster communicate operation. For inter-cluster communication, unlike TIs cross-cluster port connections, PAC DSP reutilizes the existing Memory Interface Unit (MIU) routing path to benefit load/store instructions. Programmers need to issue special pair instructions one instruction implies sending and the other implies receiving. This pair of instructions should be located in the same cycle. The purpose to reuse MIU routing path is to avoid additional circuit costs and ports to register files. However, this design result in complex restrictions because the pair instructions would occupy two slots in one cycle and only be available for B-unit and M-units. Data transfer between two local register files of I-units will be a long path. Moreover, restricted by bypassing in this design, the inter-cluster

communication will consume additional 2-cycle latency.

2.2. Software Flow

We integrate compiler, binutils, debugger, and simulator on Eclipse which was an integrated development environment (IDE) platform and used to provide a convenient development environment for application developers. The integrated software solution is shown in Figure 3.
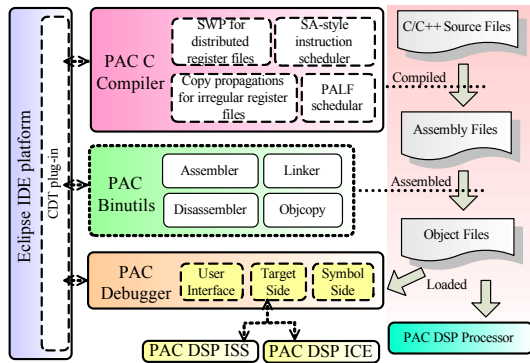

Figure 3. Software flow for PAC platforms

A graphical user interface (GUI) integrates the representation of variables, disassembly, registers, and breakpoints/watchpoints information. Users can develop programs through IDE on PAC platforms. With the support of C development tools (CDT) plug-in, Eclipse [6] integrates compiler, binutils, debugger, and simulator/emulator in a user interface. The compiler invokes the assembler and linker to produce the final ELF executable file which can run on PAC. Programmers can also fetch information, such as variables and registers, from the debugger to identify bugs of programs.

3. Compiler

The compiler for PAC DSP is developed based on ORC which is an open-source compiler infrastructure released from Intel and incorporates most of the optimization techniques of industry strength so far. This compiler is capable of generating codes with good performance on its original IA-64 target by utilizing numbers of EPIC/VLIW architectural advantages. The preliminary employment from original IA-64 to PAC DSP includes the new implementation of machine description tables and the essential supports for PAC DSP code generation. Some optimization phases, such as copy propagations for irregular register files, code generators for PAC assemblies, SA-style instruction scheduler and PALF scheduler, are published into papers [7–9]. Till now, our development of compiler with software pipeline

optimization support for PAC DSP is available. In this section, we focus on the studies of supporting basic VLIW compiler infrastructures for PAC DSP processors in SWP optimization.

3.1. Overview of PAC DSP complier

The compilation starts with processing by the frontends, generating an intermediate representation (IR) of the source program, and feeding it in the back-end. The IR , called WHIRL, is a part of the Pro64 compiler released by SGI [10]. It includes five representation levels from "very high" to "very low". Each level is invoked at the back-end to perform a series of lowering processes and optimizations on the WHIRL IR. In practice, the optimization phases are organized as a dynamically-shared library, loaded and executed on demand by the back-end. The ones who use PAC DSP compiler can turn on/off different optimization phases. These optimizations in levels are inter-procedural analysis/optimizer, loop nest optimizer, global optimizer, and code generator.

The code generator would take over the progress after the lowering and optimization phases, translating the WHIRL IR into CGIR (Code Generation Intermediate Representation), which is a low level IR reflecting the instruction set architecture of PAC DSP processor. Global and local register allocation, and final assembly codes emitting are performed here. Moreover, the code generator also performs many target dependent optimization phases, including control flow optimization, extended block (peephole) optimizer, integrated global/local scheduling (IGLS), hyperblock formation, CG loop analysis and transformation, and software pipelining.

3.2. SWP optimization for PAC

Compiler techniques, such as SWP and global instruction scheduling, have been proven to be necessary and effective methods to increase the degree of ILP in programs. The original implemented software pipeline follows these papers which wrote by Richard Huff and B. R. Rau [11,12]. Unfortunately, they did not mention about how to deal with such highly-partitioned register file architecture.

Provided software pipeline optimization to PAC DSP architecture includes many works to do. The architecture features different between IA-64 and PAC listed in Table 1. Each of them will be described separately.

PAC DSP is cluster architecture but IA-64 is not. Thus the cluster assignment phase and inter-cluster communication instruction insertion needs to be implemented into compiler. The goal of cluster assignment phase is partition loop into two sets to execute on different cluster without increasing minimal initial interval (MII) which produced by rela-

Table 1. IA-64/PAC architecture comparison

Hardware features	IA-64	PAC	
Multiple Clusters	no	yes	Cluster assignment and Intercluster communication
Distributed Register Files	no	yes	Register bank assignment
Ping-Pong Architecture	no	yes	Ping-pong constraint
Rotating Register Files	yes	no	Modulo variable expansion
Predication Support	yes	yes	Reuse the methods in ORC

tionship of dependence and utilization of functional unit. Each communication instruction insertion causes 3 cycles delay and one for instruction issues and two for additional instruction latency. To avoid increasing MII, these communication instructions should not be arranged in the critical path. In the other word, the instructions of critical path which causes recurrence MII (RecMII) should be arranged into one cluster.

To deal with PAC's hardware features on distributed register files, the register bank assignment is necessary. Considering these different types of register file, global and local, in one cluster, what kind of register file should be assigned to instructions is a problem. If data transfer between two instructions which executed by different functional unit, M-unit and I-unit, in one cluster is needed, then global register file will be assigned to them. However, such arrangement induced new problem by global register, i.e. ping-pong constraint. According to the ping-pong constraint, additional phase to insure that instruction scheduler will not arrange instructions into the worst case, such as following example, is constructed by investigators. The motivating example illustrates how the ping-pong constraint damages performance.
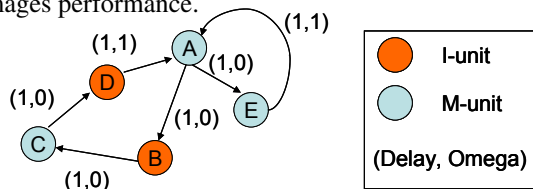

Figure 4. Motivating example

Figure 4 is a data dependence graph (DDG) built by our compiler. The orange nodes in Figure 4 executed by I-unit

and the blue nodes executed by M-unit. Delay indicates how many cycles should wait for data ready and Omega represents dependence between different iterations. Due to the special register file organization, the data-transfer between M-unit and I-unit is performed by ping-pong bank switch. In this motivating example, the result of instruction A is used by instruction B, such that register allocator allocates global register to instruction A. In this case, instruction B and E should be schedule into one cycle without any constraint. However, ping-pong constraint restricts such arrangement because of M-unit and I-unit can not access the same register file in single cycle. The code generator in compiler could get MII when it performed software pipeline optimization. MII of this DDG is dominated by RecMII and its value is 4 without any constraint. Since ping-pong constraint existed, instruction B and E could not be arranged into the same cycle. The worst case is that instruction E be arranged in first place, then instruction B. Now, MII calculated by compiler is 5 which is showed in Figure 5. The result of this example indicates that ping-pong constraint might damage the overall performance.
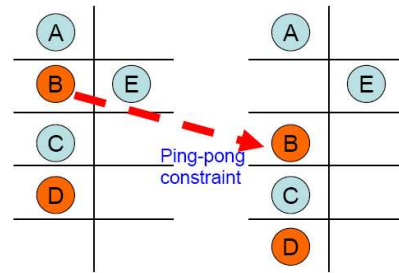
Figure 5. Scheduling result for motivating example

Our new decision phase is built into SWP to solve this problem. This decision phase identify which node should be arranged first when the ping-pone constraint occurred. In this example, ping-pong constraint occurs between instruction B and E, therefore the decision phase makes sure that instruction B would be arrange before instruction E. Unfortunately, in some situation increasing MII is necessary with ping-pong constraint, shown as Figure 6.
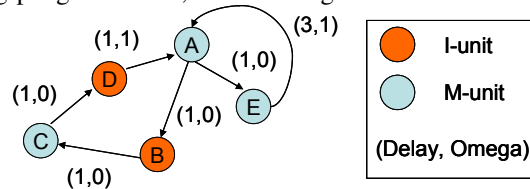
Figure 6. Situation for increasing MII

Again, the result of instruction A is used by instruction B which belong to different execution functional unit, such

that register allocator allocates global register to instruction A. In this situation, no matter which instruction, B or E, be scheduled first, then the MII will be increased.

Moreover, IA-64 has hardware support with rotating register files for performance improvement on SWP but PAC has not. Therefore, modulo variable expansion also needs to build in compiler. The rotating register files solve the lifetime overlap problem of variables. So far PAC DSP does not provide such hardware support; hence unrolling and renaming are used to handle this problem.

4. Toolkits development for PAC

In this section, we will briefly describe our experiences on development toolkit chains to this specific architecture.

4.1. Assembler/Linker

The PAC DSP Binary Utility is the collection of binary/object file tools for users who would like to build or manipulate PAC DSP executables. The collection contains assembler, linker, disassembler, and other object file manipulation tools. The compiler invokes the assembler and linker to produce the desired ELF (Executable and Linking Format) execution files.

The binary format of PAC DSP instructions adopt variable length encoding, which is an efficient strategy to build compact binary representation corresponding to different assembly code. The variable length encoding tends to encode operators, registers, immediate values, and any other meaningful data into bit fields with minimum bytes. The code size would be saved by using shorter encoding length. Therefore, we prefer to encode instructions according to their frequencies of appearance; the more frequent instructions use the shorter encoding length, and vice versa.

In addition to support variable length encoding in PAC DSP Binary Utility, we also develop some schemes to make more improvement on reducing code size. The major idea of our scheme is to compress immediate value in binary format. Consider that even if we reserve a 32-bit field in binary format for an immediate value, it is not necessary to encode immediate value with complete 32 bits. The basic compressing concept is presented as Figure 7.

Since PAC DSP hardware is able to fetch the immediate value and automatically manages signed extension to a 32-bit data, PAC DSP Binary Utility would try to rebuild the original machine encoding by compressing the immediate value with less bytes. Each machine encoding in object file is rescaned to find out if there is possibility to compress immediate value. If the immediate value in the machine encoding can be compressed, PAC DSP Binary Utility could compress the encoding length to be a shorter one. There are several steps that help us to do compressing. First, taking
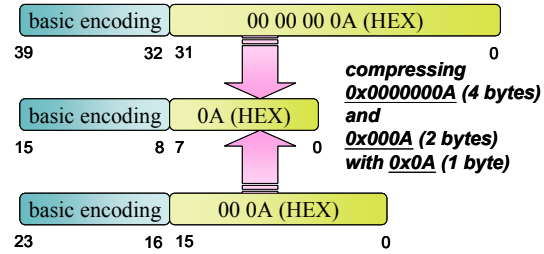

Figure 7. Compressing immediate value

the most significant bit of immediate value as consideration to identify whether it is signed or unsigned value. Second, we count the number of signed/unsigned bit in the original value encoding and discard extra bits to form a shorter binary representation for the immediate value. Next, determining the minimum bytes which could represent the value in terms of revised immediate value length and original signed/unsigned expression, taking notice of that the hardware is signed-extension. Finally, the basic encoding of instruction and the new immediate binary representation would be recombined into one single binary machine encoding. Following the steps above, it is natural to examine all compressible value of machine encodings in object file and entirely reduce code size of a application.

Another scheme to make variable length encoding better is ignoring the predicate bit field. According to PAC DSP instruction set architecture, almost all instructions are predicatable. It implies that the predicate information should be encoded into machine code. However, within a practical application, not every instruction needs to be marked as predicated. For those instructions which will be certainly executed, we can even discard their predicate information. This also saves a little code size.

To summarize our works on PAC DSP Binary Utility, not only do we deal with the complicated encoding introduced by PAC DSP architecture, but also provide more advanced code size reducing strategies.

4.2. Debugger

To provide an efficient support for application developers of PAC DSP in finding and reducing bugs or defects inside programs, GNU debugger (GDB) is employed as a basis for the debugging environment. In general, GDB consists of three major subsystems: user interface, symbol handling (the symbol side), and target system handling (the target side) [16]. The target side is an architecture-related component, which provides GDB with the common functions to access targets.

As illustrated in Figure 8, the PAC DSP GDB handles debug information of object files generated by the PAC DSP compiler in the symbol side, and communicates with in-

struction set simulator (ISS) or in-circuit emulator (ICE) via GDB Remote Serial Protocol (GDB RSP) [17]. The GDB RSP defines the rules governing the communication between debuggers and targets. In the PAC DSP GDB, a reduced and integrated GDB RSP for both ISS and ICE is proposed to control targets. For instance, we define that the memory address should be aligned in one word while accessing memory. Various protocols to set breakpoints and watchpoints in the debugging programs are also defined. The modifications in our PAC GDB RSP decrease the complexity of designing the remote stub for both PAC DSP ICE and PAC DSP ISS. Beside physical registers, the PAC GDB RSP defines pseudo registers to transfer system information such as interrupts, timers, and control registers. By using pseudo registers, users can inspect sufficient information of the system in run time.

For source level debugging, a translation from the register numbering used by the PAC DSP compiler to the numbering defined in the PAC GDB RSP is provided in the target system handling subsystem. The translation bridges the gap between the debug information and the PAC GDB RSP to guarantee the correctness of the collected information.
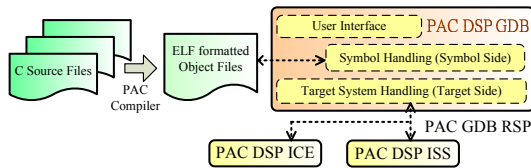

Figure 8. The PAC GDB setup

5. OS support

pCore (passive/compact runtime environment) microkernel operating system is designed for PAC VLIW DSP processors under the dual-core/multi-core scenario which tries to provide a minimal but sufficient OS services with a small kernel. It is a preemptive, multitasking and static microkernel which cooperates with the embedded OS that runs on the main processor in a dual-core/multi-core platform that allows application developers to build the software infrastructure of PAC DSP processors. Figure 9 shows the architecture overview of pCore microkernel which is basically composed of resource management, synchronization modules, inter-process communication and some kernel services.

The kernel structure defines a task as a running program on pCore which is managed by the kernel. The kernel maintains the runtime information for each task by building the task control block, TCB, to keep the necessary information. The data structure of TCB in pCore contains no indirect pointers to make the access to the data fields more
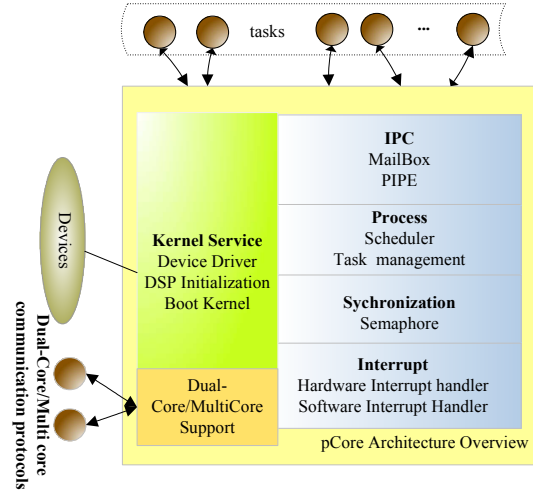

Figure 9. pCore architecture overview

efficiently. Furthermore, the size of memory required for TCBs is configurable when the running applications are determined. pCore adopts a priority-based scheduling algorithm to make the whole system deterministic and allow the master processor to have a better control on it under the master-slave programming model. The communication between tasks in pCore bases on the message-passing scheme which constructs a message as a simple data structure that contains an identifier of the sender and receiver, a pointer to a data, a field of data and some useful information. Two basic mechanisms, PIPE and MAILBOX based on the message-passing scenario are provided for the programmer to perform interprocess communication.

In a multi-tasking system like pCore, the synchronization between tasks is required in order to ensure the correctness of concurrent access to data. pCore supports a mechanism for creating critical sections by using counting semaphores which are implemented by disabling the interrupts. As the target processor, PAC DSP processors are intended to be employed in the dual-core/multi-core platform. pCore provides a friendly and efficient system calls for the programmer to register their own service routines. This system calls allow the programmer easily migrating to any customized dual-core/multi-core communication protocol on it.

pCore is a tiny microkernel based OS which is designed for PAC DSP processors with a very small kernel size, nevertheless, it implements only a minimal set of services such as inter-process communication, scheduler and some other kernel services. Furthermore, it supports the dual-core/multi-core programming model by providing necessary system calls for the users to port any customized dual-core/multi-core communication protocol on it.

6. Results

Preliminary experiments were done by running the DSP-stone benchmark suite [20] on the PAC DSP. Since the PAC DSP compiler is still in progress, we only evaluated optimizations in O1 level for the early-stage performance evaluations of our designs. Some programs of DSP-stone benchmark suite will not be processed by SWP optimization because their trip counts of loops are too small to gain benefit. Table 2 shows the cycle counts for some programs of DSP-stone benchmark suite estimation on PAC ISS. The better results are always produced by code generation with SWP.

Table 2. Cycle counts of DSP-stone benchmark suite

|	Without SWP	With SWP
fir	2672	1208
lms	3343	1865
fir2dim	14042	10636
matrix1	71467	36739
matrix2	68671	34363
mit1x3	681	480
n-real updates	3292	2153
n-complex updates	6792	3145
convolution	1524	773
dot of product	258	178
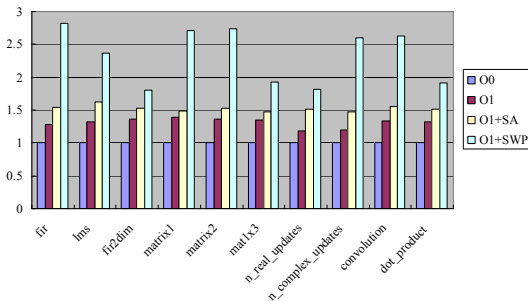
Figure 10. Performance result

Experimental results of the DSP-stone benchmark suite running on the PAC DSP are normalized and presented in Figure 10. Blue bar showed the performance generated from the compiler without any optimization and purple bar showed the performance generated from the compiler with some optimization in O1, such as EBO and WOPT. Yellow bar showed the performance got from our previous works on register allocation which uses the simulated annealing (SA) method to improve local instruction scheduling. Green bar showed the performance got from the compiler with SWP

optimization. These results indicated that our SWP policies gain performance improvement around 2.5 times against to the code generated without any optimization. Clearly, software pipelining is also an effective method to attain performance on highly partitioned register files of PAC architecture. However, we noticed that utilization of resources on the PAC architecture is low when we investigated the code which was scheduled by SWP optimization. In the other word, most of the instructions will be arranged into one cluster. Due to the characteristic of DSP-stone benchmarks, while performing SWP optimization on these programs, most of the MII were dominated by their RecMIIs. Therefore, such instruction arrangement was done by cluster assignment phase with the goal to minimize the initial interval. Hence, in our future works we will try to boost the utilization rate of resources. The high-level loop transformations should also be used to enhance partitioning scheme and improve the performance.

In addition to the significant improvement of SWP optimization, the current implementation of pCore for PAC DSP processor is proved to be very tiny with the size less than 10K Bytes. The kernel is impressively small comparing with many existing embedded microkernel OS or real-time kernels. Moreover, by configuring the kernel before runtime when the applications running on it are determined, the memory requirement of pCore could be shrank even more. Figure 11 shows the module size distribution
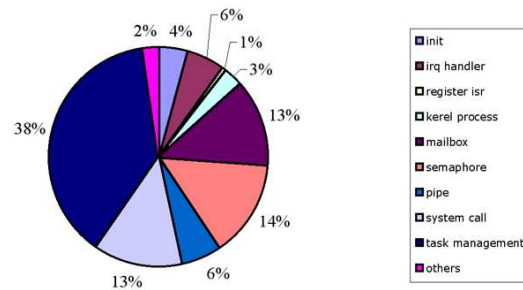
Figure 11. Module size distribution of pCore

of pCore which presents the result of implementation. The task management module which is the key feature of multi-tasking turns out to have a large memory requirement. If the DSP is employed to run one application at a time, the kernel can be configured to be single-tasking mode which can reduce the memory requirement of the kernel in about 50%.

Furthermore, those toolkits we developed , such as compiler, assembler/linker and debugger, are integrated into Eclipse to provide a convenient environment for application developers. As shown in Figure 12, the graphical user interface (GUI) of our PAC DSP debugger could offer the visual representation of variables, disassembly, registers, and

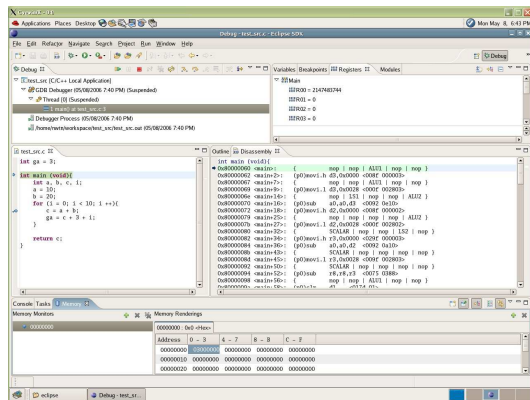breakpoints/watchpoints information in an integrated style.

Figure 12. A snapshot of Eclipse for PAC DSP

7. Conclusion

This article attempted to describe our experiences on development OS and toolkit chains on PAC architecture with strict constrains on register files access. We developed a tiny microkernel based OS which is designed for PAC DSP processors with a very small kernel size. This OS implements a minimal set of services and supports the dual-core/multi-core programming model by providing necessary system calls for the users to port any customized dual-core/multi-core communication protocol on it. We also built assembler/linker to do the basic machine encoding and provide more advanced code size reducing strategies on PAC. Besides, we provided a debugging environment and integrated software solutions for application developers of PAC DSP. Furthermore, we introduced our experiences on retarget software pipeline optimization from IA-64 to PAC DSP. The experimental evaluation using DSP-stone benchmark indicates significant improvement of cycle time. These experiences might benefit the architecture designers and compiler developers who are interested in similar heterogeneous clustered VLIW architectures with port-restricted, distinct partitioned register file structures.

8. Acknowledgements

References

[1] 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.

[2] 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

[3] T.-J. Lin, C.-C. Chang. C.-C. Lee, and C.-W. Jen An Efficient VLIW DSP Architecture for Baseband Processing. In *Proceedings of the 21th International Conference on Computer Design*, 2003.

[4] 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. In *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, April 2005.

[5] Roy Ju, Sun Chan, and Cheng yong Wu, "Open Research Compiler for the Itanium Family", *Tutorial at the 34th Annual Intl Symposium on Microarchitecture,* Dec, 2001.

[6] Eclipse Platform Technical Overview, International Business Machines Corp., 2001. http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf

[7] Yung-Chia Lin, Chung-Lin Tang, Chung-Ju Wu, Ming-Yu Hung, Yi-Ping You, Ya-Chiao Moo, Sheng-Yuan Chen, Jenq-Kuen Lee. "Compiler Supports and Optimizations for PAC VLIW DSP Processors", *LCPC*, 2005.

[8] Yung-Chia Lin, Yi-Ping You, Jenq-Kuen Lee. "Register Allocation for VLIW DSP Processors with Irregular Register Files", *CPC*, 2006.

[9] Cheng-Wei Chen, Yung-Chia Lin, Chung-Ling Tang, Jenq-Kuen Lee. "ORC2DSP: Compiler Infrastructure Supports for VLIW DSP Processors", *IEEE VLSI TSA*, April 27-29, 2005.

[10] SGI - Developer Central Open Source - Pro64 http://oss.sgi.com/projects/Pro64/

[11] Richard Huff, "Lifetime-Sensitive Modulo Scheduling", *PLDI SIGPLAN*, 1993.

[12] B. R. Rau, M. Lee, P. P. Tirumalai, M. S. Schlansker, "Register Allocation for Software Pipelined Loops", *ACM SIGPLAN*, 1992.

[13] A. Capitano, N. Dutt, and A. Nicolau, "Partitioned register files for VLIWs: A preliminary analysis of tradeoffs", *Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO-25),* Portland, December 1V4, 1992; pages 292V300.

[14] Texas Instruments: TMS320C64x Technical Overview. Texas Instruments, Feb 2000.

[15] The open research compiler official page. http://ipf-orc.sourceforge.net/.

[16] John Gilmore and Sten Shebs, "GDB Internals: A guide to the internals of the GNU debugger" *Free Software Foundation, Inc*, 2006.

[17] Bill Gatliff, "Embedding with GNU: the gdb Remote Serial Protocol" *Embedded System Programming*, pp.108-113, November, 1999.

[18] Tay-Jyi Lin, Chen-Chia Lee, Chih-Wei Liu, and Chein-Wei Jen "A Novel Register Organization for VLIW Digital Signal Processors", *Proceedings of 2005 IEEE International Symposium on VLSI Design, Automation, and Test*, 2005, pages 335 V338.

[19] R. Leupers "Instruction scheduling for clustered VLIW DSPs", *Proc. Intl Conference on Parallel Architecture and Compilation Techniques,* Oct. 2000, pages 291V300.

[20] V. Zivojnovic, J. Martinez, C. Schlager and H. Meyr "DSPstone: A DSP-Oriented Benchmarking Methodology", *Proc. of ICSPAT*, Dallas, 1994.