Shared-Port Register File Architecture for Low-Energy VLIW Processors

NEERAJ GOEL, ANSHUL KUMAR, and PREETI RANJAN PANDA,

Department of Computer Science and Engineering, IIT Delhi, India

We propose a reduced-port Register File (RF) architecture for reducing RF energy in a VLIW processor. With port reduction, RF ports need to be shared among Function Units (FUs), which may lead to access conflicts, and thus, reduced performance. Our solution includes (i) a carefully designed RF-FU interconnection network that permits port sharing with minimum conflicts and without any delay/energy overheads, and (ii) a novel scheduling and binding algorithm that reduces the performance penalty. With our solution, we observed as much as 83% RF energy savings with no more than a 10% loss in performance for a set of Mediabench and Mibench benchmarks.

Categories and Subject Descriptors: C.1.1 [Processor Architectures]: Single Data Stream Architectures; D.3.4 [Programming Languages]: Processors; B.3.2 [Memory Structures]: Design Styles

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Low energy, register file design, VLIW architecture

ACM Reference Format:

Neeraj Goel, Anshul Kumar, and Preeti Ranjan Panda. 2014. Shared-port register file architecture for low-energy VLIW processors. *ACM Trans. Architect. Code Optim.* 11, 1, Article 1 (February 2014), 32 pages. DOI: <http://dx.doi.org/10.1145/2533397>

1. INTRODUCTION

Very Long Instruction Word (VLIW) processors such as Trimedia [van Eijndhoven et al. 1999], TI's TMS3206x [Seshan 1998], and ST's Lx [Faraboschi et al. 2000] are a class of ILP (instruction-level parallel) architectures where the order of instruction execution is determined at compile time. These architectures are mostly used in DSP, media, and embedded applications. In VLIW processors, multiported Register Files (RFs) consume a significant part of the total energy. A study performed on the Trimedia processor [van de Waerdt et al. 2005] shows that 20% of the processor power is consumed in the RF. In another study, where a number of processor cores were used in building a platform, register files accounted for 37% of total power consumption [Lambrechts et al. 2005]. In this article, we investigate reduction of register file ports with an objective of reducing the overall energy consumption in a VLIW processor, without any significant performance loss.

In a VLIW processor, multiple operations are issued in one cycle, requiring parallel access to the register file. If one Function Unit (FU) requires two operands and produces one result, then for N FUs, $2N$ read and N write RF ports are provided with the RF,

---

Authors' addresses: Neeraj Goel, (Current address) Synopsys, Tower B, LogixTech Park, Sector 127, Noida, India; email: dr.neeraj.goel@gmail.com; Anshul Kumar, Department of Computer Science, IIT Delhi, Delhi, India; email: anshul@cse.iitd.ernet.in; Preeti Ranjan Panda, Department of Computer Science, IIT Delhi, Delhi, India; email: panda@cse.iitd.ernet.in.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2014 ACM 1544-3566/2014/02-ART1 \$15.00

DOI: <http://dx.doi.org/10.1145/2533397>

Fig. 1. RF read port usage.

with one-to-one connections between the FU ports and the RF ports. However, the average RF port usage per cycle (read plus write) is less than $3N$, because some of the operations such as *move* have only one operand to read, while in some other operations, immediate operands do not require a register access and store operations do not write back into the RF. Moreover, the fact that the average Instructions Per Cycle (IPC) for an application is less than the peak IPC also contributes to lower average usage of RF ports per cycle. Further, as shown by Goel et al. [2007], Park et al. [2002], and Sami et al. [2002], RF read is not required for operand read from the bypass network. Bypass network, also known as forwarding paths, is a standard architectural technique implemented to avoid data hazards in processor pipelines. RF writes can also be avoided for those values that can be read from the bypass network and are not required for future instructions.

Figure 1 shows the effect of these factors on average RF port usage for a set of high ILP benchmarks. The benchmark set consists of a number of kernels and applications from the embedded systems domain with high ILP (discussed in more detail in Section 6.1). These benchmarks were executed using the Trimaran compiler framework [Chakrapani et al. 2005] targeting the HPL-PD VLIW processor architecture [Kathail et al. 2000] with varying numbers of FUs. The figure shows the difference between peak read-port requirement and port requirement due to average parallelism (given by $2 \times$ average IPC). Average read-port usage is even lower due to fewer ports required by certain operations. Average RF port usage further reduces with operands available in the bypass network. For example, for a six-issue processor, on average, three read ports are used, though the maximum port usage is 12 read ports. Provisioning 12 read ports in this case would be a waste of available RF bandwidth. Moreover, the cost of number of ports is quite high as the RF power increases superlinearly (N^2 to N^3) with the number of ports [Zyuban and Kogge 1998].

With this motivation, in this article we propose a reduction in the number of RF ports by sharing them among FUs. However, reduction of ports may lead to conflicts (we call these *port conflicts*), causing an increase in the execution time. In the shared-port RF architecture, RF-FU connections are no longer one to one. Depending on the structure of the RF-FU interconnection network, there may be additional conflicts (we call these *path conflicts*) due to lack of interconnection paths, even though ports may be available. In this article, we present an approach to keep both these kinds of conflicts within acceptable levels while retaining the benefits of energy savings.

We examine different RF-FU interconnection options and propose a simple and generic (direct) interconnect topology that has minimal multiplexing overheads, has low power consumption, yet is performance effective. We present scheduling and binding algorithms that are aware of RF port and interconnection constraints and minimize the increase of schedule length by reducing the port and path conflicts. Our approach shows 73% reduction in the RF energy with less than 6% performance loss for a set of Mediabench and MiBench benchmarks.

Area and access time of an RF also increase superlinearly with number of ports [Rixner et al. 2000]. Therefore, along with energy savings, shared-port RF also leads to faster RF access and reduced area [Park et al. 2002; Kim and Mudge 2003], though in this article, we focus on the energy aspect of shared-port RF architecture.

The rest of this article is organized as follows. Previous work is presented in Section 2. Performance and energy models of the reduced-port RF architecture are discussed in Section 3. Sections 4 and 5 discuss architecture and code generation details of the proposed scheme. Experimental results are discussed in Section 6, and finally our conclusions are presented in Section 7.

2. PREVIOUS WORK

In a single-issue processor, RF port sharing among multiple function units is a matter of rule, but shared-port RF architectures for multiple-issue processors have received very little attention. In VLIW processors, typically, port sharing has been used at the level of issue slots; that is, function units in a single issue slot share the same read and write ports [Seshan 1998]. Our approach goes one step further and permits sharing ports among issue slots. Aditya et al. [1999] proposed an approach with very limited port sharing. In their approach, an FU having a lesser port requirement shares an RF port with an FU having a higher port requirement.

There are some instances of reduced-port RFs in other multi-issue architectures. For example, some authors [Park et al. 2002; Kim and Mudge 2003; Sangireddy 2007; Sirsi and Aggarwal 2009] have suggested port sharing for superscalar processors. In superscalar architectures, port conflict management in hardware causes an increase in hardware cost. However, for a VLIW processor, a compiler-driven solution ensures that there are no port conflicts at runtime. In addition, using schedule information, the compiler optimizes the performance for a given number of read and write ports. In Transport Triggered Architectures (TTAs) [Corporaal 1999], the register file requires fewer ports than traditional operation-triggered architectures [Hoogerbrugge and Corporaal 1994]. In TTA architectures, explicit operand (data) movements and visibility of transport buses to the compiler result in a lesser number of ports in the RF. Due to the difference in architecture exposure to the compiler and limited control of instruction set on operand movements, TTA techniques (both compiler algorithms and hardware mechanisms for RF port reduction) are not directly applicable to VLIW architecture.

While this article considers port reduction in a monolithic RF, there are techniques such as clustering [Capitanio et al. 1992; Lowney et al. 1993; Ellis 1986] and banking [Cruz et al. 2000; Balasubramonian et al. 2001] that result in fewer ports per cluster or per bank, thus achieving energy reduction. Clustering and banking also reduce the complexity of the bypass network, making RF scalable for high-issue-width processors. However, these techniques have performance and energy overheads due to intercluster data movements or increased interconnection network complexity. Yet another class of approaches reported in the literature achieves RF energy reduction by reducing RF size. This includes use of multilevel RF [Zalamea et al. 2000; Cruz et al. 2000; Balasubramonian et al. 2001] and register packing [Ergin et al. 2004]. It is well understood that size reduction is less effective in RF energy reduction as compared to port

reduction [Shivakumar and Jouppi 2001]. Finally, we observe that port reduction can be used orthogonally with various other techniques mentioned here.

Bypass-aware scheduling for superscalar processors proposed by Park et al. [2006] increases the probability of reading values from the bypass network, and Goel et al. [2007] proposed a compiler-driven bypass architecture. We build on these techniques and propose reduced port-aware scheduling and binding algorithms that exploit the availability of values in the bypass network. Port binding was suggested in initial VLIW architectures (e.g., Bulldog [Ellis 1986]), but their instruction scheduling was not constrained by the number of RF ports.

In this work, we assume the complete bypass network; that is, all possible forwarding paths exist, as most of the commercial VLIW processors such as Trimedia, Lx, and Itanium follow the same bypass structure. In the past, there have been several studies to use only selected forwarding paths such that impact on performance is minimal [Fan et al. 2003; Shrivastava et al. 2005], and compiler techniques were suggested to further minimize the impact on performance [Park et al. 2006; Kudlur et al. 2004]. As less frequently used paths are not present in the limited bypass case, there will be minimal effect on RF energy savings achieved due to our proposed architecture.

3. IMPACT OF RF PORT SHARING ON PERFORMANCE AND ENERGY

In Section 1, we provided the motivation for a VLIW architecture with a reduced number of RF ports. In this section, we analyze the impact of port sharing on RF energy and performance.

3.1. RF Energy

Energy consumed by RF is the sum of the dynamic energy and leakage energy. Dynamic energy depends on activity and can be calculated by counting the number of RF accesses and energy per access. Leakage energy depends on leakage current and execution duration. Leakage current is usually fixed for a supply voltage. Execution time is the number of execution cycles multiplied by the clock period. Thus, RF energy, E_{rf} is

$$E_{rf} = E_{read}N_{read} + E_{write}N_{write} + P_{rf-leak}tC, \quad (1)$$

where E_{read} , E_{write} , N_{read} , N_{write} , $P_{rf-leak}$, C , and t are energy per read access, energy per write, number of RF reads, number of RF writes, RF leakage power, number of execution cycles, and clock period, respectively.

N_{read} and N_{write} are the number of RF reads and writes after avoiding redundant reads and writes due to operands available in the bypass network. The total number of operand reads and result writes in an application is fixed. However, operand reads from bypass paths may change with the number of read ports and write ports as the schedule gets extended with reduction in ports. Considering this change in N_{read} and N_{write} as a second-order effect, we ignore it for computation of RF energy in our model. However, we consider this effect in our experiments and evaluations.

The RF access energy for each read and write can be calculated using CACTI 4.2 [Tarjan et al. 2006]. CACTI 4.2 is an analytical model for SRAM, which estimates area, power, energy, and access time. CACTI computes read access energy as the sum of decode energy, bitline energy, wordline energy, sense amplifier energy, and output drivers energy. Write energy is computed as the sum of decode energy, bitline energy, and wordline energy. Due to sense amplifiers and output drivers, the read access energy is usually more than write access energy. Table I gives some E_{read} and E_{write} values computed using CACTI 4.2, showing variation with number of ports. The energy numbers are for a 64-bit-wide register file with 64 entries at 130nm and 70nm technology nodes. It is apparent from the table that the energy per access increases linearly with increase in the number of ports. As all read and write accesses in the RF occur in parallel, power

Table I. Read and Write Access Energy for Different Number of Parallel Read and Write RF Ports

N_{read}	N_{write}	Technology node 130nm			Technology node 70nm		
		E_{read} (pJ)	E_{write} (pJ)	$P_{rf.leak}$ (mW)	E_{read} (pJ)	E_{write} (pJ)	$P_{rf.leak}$ (mW)
12	6	14.9995	8.02577	13.5673	3.53555	1.88965	19.7591
10	5	13.3052	6.88919	10.9866	3.13669	1.62261	16.2086
8	4	12.0412	5.75047	8.5338	2.73736	1.35493	12.761
6	3	10.2388	4.60972	6.20868	2.33753	1.08658	9.41635
4	2	8.43658	3.468	4.01136	1.98981	0.817916	6.17463
2	1	6.736	2.3442	1.94179	1.58903	0.552478	3.03585

consumed by the RF increases quadratically $((N_{read} \times E_{read} + N_{write} \times E_{write})/t)$. Also, at smaller geometries, dynamic read/write energy reduces but leakage power increases.

Our architecture achieves reduction in RF energy due to three factors: (a) decrease in E_{read} and E_{write} by virtue of reduced ports, (b) decrease in N_{read} and N_{write} by virtue of the bypass network, and (c) decrease in leakage energy ($P_{rf.leak}tC$); with the port reduction, the number of execution cycles C increases marginally, but decrease in $P_{rf.leak}$ is significant (as indicated from Table I).

We now study the effect of port reduction on the number of execution cycles, C .

3.2. Performance

For an N issue processor, the unconstrained RF has $2N$ read ports and N write ports. If read and write ports of the register file in such a processor are limited to k and m , respectively, the instructions that require more than k read ports or m write ports need to be rescheduled such that port constraints are met. The number of cycles required for scheduling such instructions due to read-port constraint can be estimated as the total number of reads in these instructions divided by k . Similarly, the number of cycles due to write constraints can be estimated. Consider a simulation trace snippet in which 10 cycles require five read ports and five cycles require six read ports. If the number of read ports is constrained to four, the number of estimated cycles required is $(10 \times 5 + 5 \times 6)/4 = 20$, instead of 15.

For this computation, we use two vectors $\mathbf{R}$ and $\mathbf{W}$ of length $2N$ and N , respectively. R_i (or W_i) denotes the number of cycles using i read ports (or i write ports) for execution of an application on the architecture with the unconstrained RF ports. The number of additional cycles due to read-port constraint ($Cycle_{read}^+$) and write-port constraint ($Cycle_{write}^+$) is the difference between estimated cycles and original cycles requiring ports more than k and m , respectively:

$$Cycle_{read}^+ = \frac{\sum_{i=k+1}^{2N} R_i \times i}{k} - \sum_{i=k+1}^{2N} R_i. \quad (2)$$

$$Cycle_{write}^+ = \frac{\sum_{i=m+1}^N W_i \times i}{m} - \sum_{i=m+1}^N W_i. \quad (3)$$

The additional cycles due to both read-port constraint and write-port constraint is estimated as:

$$Cycle^+ = Cycle_{read}^+ + Cycle_{write}^+. \quad (4)$$

The total number of execution cycles is:

$$C = \sum_{i=0}^{2N} R_i + Cycle^+, \quad (5)$$

Fig. 2. Performance impact due to shared RF ports.

where the first term gives the total number of execution cycles in case of unconstrained port execution.

A good scheduler attempts to match the behavior expressed in Equations (2)–(5). Therefore, for an application, Equation (5) can be used to find the approximate execution cycles for same-issue-width processors with any number of read and write ports in the RF, with characteristic vector $\mathbf{R}$ and $\mathbf{W}$ computed once. Note that this performance model is an approximation of the actual scheduling process and can overestimate or underestimate performance. The actual performance depends on several other factors, such as data, control, and memory dependency in the application; quality of scheduling and binding; interconnection topology for the shared-port RF; and so forth.

We conducted an experiment on a set of high ILP benchmarks to observe the performance impact due to shared-port RF architecture as estimated by the previously mentioned model. Characteristic vectors $\mathbf{R}$ and $\mathbf{W}$ were obtained by executing benchmark applications for a four-issue processor with unconstrained RF (with eight read and four write ports) using the Trimaran simulator. The execution cycles for other RF configurations were estimated using Equation (5) and normalized with respect to the eight read and four write RF configuration. Figure 2 shows the normalized average execution cycles estimated by the model for different RF configurations. In the figure, we refer to an RF configuration by the number of read and write ports; for example, the 4r3w configuration represents a shared-port RF with four read ports and three write ports.

From these results, it can be seen that unless the reduction in the number of ports is large, the effect on performance is marginal. For example, in case of a four-read-port and three-write-port RF, using the model, we observe an increase of 5% in execution cycles. These observations suggest that the reduced-port RF architecture is potentially energy saving and has low performance overheads. This motivates us to further explore the design aspects of this architecture and the compiler requirement for the same. We discuss the design aspects of the architecture in the next section.

Fig. 3. RF-FU interconnection topologies for shared-port register file.

4. PROCESSOR DESIGN WITH SHARED-PORT RF

With the shared-port register file, only register read and write stages of the processor pipeline are modified, while the rest of the processor design remains unaltered. In the conventional VLIW architecture, FU and RF ports have one-to-one connectivity. In commercial architectures such as *velociTI* [Seshan 1998] and *Lx* [Faraboschi et al. 2000], the RF ports are shared by a number of function units in the same issue slot. The function units in a particular issue slot form a composite unit, which can be viewed as a single FU at the architectural level.

In the proposed shared-port RF architecture, one RF port may be connected to more than one FU port. Therefore, an interconnection network is required to map an RF port to an FU port and operand address to an RF address port. The mapping logic and the access energy of the RF read/write depends on the RF-FU interconnection network.

4.1. RF-FU Interconnection Network

We classify different interconnection topologies into three classes: complete, partial, and direct. The complete interconnection networks form one extreme, in which every FU port is connected to every RF port (Figure 3(a)). On the other extreme are direct interconnection topologies in which each FU port is connected to only one RF port (Figure 3(b)). There are numerous partial interconnection topologies in between these two extremes (Figure 3(c)). The direct interconnection networks require the least multiplexing and, therefore, offer the least RF access energy and access delay, though having maximum possibility of path conflicts. On the other hand, the complete interconnection eliminates path conflicts by providing all possible interconnection paths, thus offering the best performance, ignoring delays caused by the interconnection network itself. The connections shown in Figure 3 are between RF read ports and FU inputs. A similar set of connections exist between RF write ports and FU outputs.

Due to RF port sharing, each FU read and write port is mapped to a specific port (or ports) of RF. We define this mapping of FU read/write port to RF port as *port mapping*. In the direct interconnection, each FU port is connected to a single RF port; therefore, FU bindings determine the RF port mapping. In case of complete interconnection, port-mapping information may be generated by the compiler and passed to the hardware, but it requires additional bits in the instruction. The additional bits will result in an increase in code size and significant modification in the fetch stage of the processor. The alternative is to perform port mapping in the hardware. The compiler, in this case, just ensures that the number of RF reads and writes in a cycle is less than or equal to the number of available RF ports.

For partial interconnection, the port-mapping problem is nontrivial. It can be mapped to the maximal flow problem [Papadimitriou and Steiglitz 1998] and solved optimally at compile time, but will require additional instruction bits to pass the information. Solutions are feasible in hardware also, but only suboptimal solutions are practical in hardware. Moreover, the compiler still has to generate an FU binding for which the hardware will be able to find a port mapping that is free from path conflicts. For this, the compiler would need to know the exact hardware algorithm. To avoid such a tight coupling between hardware implementation and compiler implementation, we do not consider partial interconnection further.

In multi-issue processors, different issue slots are usually homogeneous for most frequent operations, while less frequent operations are available on fewer-function units. For example, in Lx architecture, integer operations can be performed by all FUs, while memory operations can be performed by one in four FUs. Considering the homogeneity of FUs, a careful selection of the direct interconnection matrix makes more FUs available as binding options to the scheduler. Therefore, the path conflicts can be kept low, resulting in a performance close to that of complete interconnection while retaining the advantages of the simple hardware and multiplexer-less connections.

4.1.1. Choosing the Direct Interconnection Matrix. The direct interconnection structure can be represented by a matrix P . For example, $P_{ij}^1 = 1$ and $P_{ij}^2 = 0$ mean there is a path from the i^{th} RF read port to the first input of FU_j , but no path from the i^{th} RF read port to the second input of FU_j . $P_{ij}^w = 1$ if there is a path from the i^{th} write port of the RF to the output of FU_j .

There are M^L possible direct interconnection matrices, where L and M are the number of FU ports and RF ports, respectively, considering read and write ports as independent. M^L also includes the matrices that use less than M RF ports. In an RF, all ports are symmetrical and every pair is equivalent. This property of RF reduces the number of candidate interconnection matrices significantly. The number of unique interconnection matrices for a given M and L is an order of magnitude less than the total number. Further, we have defined the following guidelines to narrow down the choice:

- (i) The two read ports of each FU are connected to different RF ports. For example, as shown in Figure 4(a), if both ports of an FU are connected to the same RF port, that FU will not be able to access two distinct operands at same time. Therefore, this is a necessary condition for a valid interconnection network.
- (ii) Assuming that most frequent operations can be executed by all FUs of the processor, each RF port should be connected to approximately an equal number of FU ports such that the resulting interconnect is balanced. A balanced interconnect distributes the chances of port conflict uniformly across all FUs. For example, as shown in Figure 4(b), RF port 1 and 2 are not shared, while port 3 and 4 are

Fig. 4. Example direct interconnects and corresponding interconnection matrices.

shared among three FUs. Such an imbalanced interconnection network leads to more path conflicts and fewer operations can be bound to FUs in a cycle.

- (iii) Typically, the first port of an FU is used to read operands from the RF more often than the second port. This happens because of single operand instructions (MOVE, LOAD, INC, etc.) and instructions with immediate operands. For example, if there are three operations, MOVE, LOAD, and ADD, and the direct interconnection matrix as shown in Figure 4(c) is used, only two operations can be bound in the same cycle due to path conflicts. On the other hand, if read ports of FU1 are connected to RF ports 1 and 2, and read ports of FU2 are connected to RF ports 2 and 1, then both MOVE and LOAD could be bound to FU1 and FU2, respectively, and ADD could be bound to FU3. Therefore, for fewer path conflicts and for more balanced sharing, we require that an RF port be shared with the first read port of one FU and the second read port of another FU. In other words, first and second FU read ports should be separately distributed among RF ports as uniformly as possible.

Mathematically, these conditions may be written as

- (i) $P_{ij}^1 \cdot P_{ij}^2 \neq 1; \forall i, j$.
- (ii) $\lfloor 2N/k \rfloor \leq \sum_j (P_{ij}^1 + P_{ij}^2) \leq \lceil 2N/k \rceil; \forall i$ and $\lfloor N/m \rfloor \leq \sum_j P_{ij}^w \leq \lceil N/m \rceil; \forall i$, where N , k , and m are the number of FUs, RF read ports, and RF write ports, respectively.
- (iii) $\lfloor N/k \rfloor \leq \sum_j P_{ij}^1 \leq \lceil N/k \rceil; \forall i$ and $\lfloor N/k \rfloor \leq \sum_j P_{ij}^2 \leq \lceil N/k \rceil; \forall i$.

In an interconnect configuration, if an RF port is connected to more (or less) FU ports than in the balanced interconnection defined previously, the difference is called the *port imbalance factor* for the port. The total RF port imbalance factor is calculated as the sum of the port imbalance factors of all RF ports, that is,

$$\begin{aligned}
 \text{Port imbalance factor} = & \text{MAX} \left(\lfloor 2N/k \rfloor - \sum_j (P_{ij}^1 + P_{ij}^2), \sum_j (P_{ij}^1 + P_{ij}^2) - \lceil 2N/k \rceil \right) \\
 & + \text{MAX} \left(\lfloor N/m \rfloor - \sum_j P_{ij}^w, \sum_j P_{ij}^w - \lceil N/m \rceil \right) \\
 & + \text{MAX} \left(\lfloor N/k \rfloor - \sum_j P_{ij}^1, \sum_j P_{ij}^1 - \lceil N/k \rceil \right) \\
 & + \text{MAX} \left(\lfloor N/k \rfloor - \sum_j P_{ij}^2, \sum_j P_{ij}^2 - \lceil N/k \rceil \right).
 \end{aligned}$$

Fig. 5. Example direct interconnects and corresponding interconnection matrices.

Port imbalance factor gives the extent of deviation from a balanced interconnection network.

It turns out that the number of matrices that satisfy these guidelines is very small. The total number of possible matrices is M^L , and the number of those using all M RF ports, $F(M, L)$, is defined by the following recurrence: $F(M, L) = M^L - M \times F(M - 1, L)$; $F(1, L) = 1$. The number of unique interconnection matrices, taking into account symmetry of the RF ports, is given by the number of ways to partition a set of L elements into M nonempty subsets. This number is also called the *Stirling number of second kind* $S(L, M)$ [Graham et al. 1988]. The number of interconnection matrices that follow the first, second, and third guideline can be found by enumerating the subsets given by Stirling numbers of the second kind and applying the given constraints.

For example, for 8 FU ports and 4 RF ports, 65,536 interconnection matrices are possible, out of which 42,320 use all 4 RF ports and only 1,701 ($S(8, 4)$) are unique. The number of matrices that follow (i) first; (ii) first and second; and (iii) all three guidelines are 652, 60, and 9, respectively. Any interconnect that follows the given guidelines leads to less performance impact due to path conflicts. A few examples of such interconnects are given in Figure 5. In these examples, 4 RF read ports are shared between 8 FU input ports. Each RF port is connected to two FU input ports; one of them is the first input port and the other is the second input port, to ensure homogeneity. The figure also shows the corresponding interconnection matrices. Each row of the interconnection matrix indicates the RF port to which an FU port is connected.

4.2. Bypass-Aware RF Access

The number of RF reads/writes can be reduced by using bypass-aware register file access, consequently reducing the RF port requirement. Operands read from pipeline registers using the bypass network need not be read from the register file. Further, if all reads of an operand can be done from the bypass network, then it need not be written to the RF at all. In VLIW processors, where the compiler is aware of the execution order, bypass usage can be determined at compile time and the relevant information passed on to the hardware either through extra bits for each operand [Sami et al. 2002] or by reassignment of some architecture register addresses to point to pipeline registers instead [Goel et al. 2007]. The former scheme entails a change in the instruction format, while the latter could cause a marginal performance loss. We follow the latter scheme for bypass-aware register file access because the performance loss is negligible.

It may be noted that using bypass values to avoid RF read/write requires extra logic in exception handling. Exception handling procedures need to store a few pipeline registers along with the processor state. Pipeline stalls do not affect the proposed scheme unless the values in the pipeline registers get overwritten. Preventive measures

are required to avoid overwriting of pipeline registers. For instance, pipeline stalls due to data cache or instruction cache misses may lead to flushing of the rest of the pipeline. In such cases, a signal is required that stalls all the pipeline stages after a cache miss. The pipeline is resumed when we obtain the value from the cache.

It has been discussed by Goel et al. [2007] that bypass-aware RF access leads to energy savings as well as area savings. The technique proposed in this article utilizes the bypass network to avoid RF reads and writes; the bypass network itself is assumed to be present in the architecture to avoid performance losses due to read-after-write hazards. The area savings are due to reduced complexity of control logic of the bypass network by directly addressing pipeline registers.

5. CODE GENERATION FOR REDUCING PORT AND PATH CONFLICTS

To produce correct and optimized code, the compiler plays an important role associated with the proposed architecture. The compiler's scheduling and binding algorithms ensure correctness by preventing port conflicts and path conflicts during execution. These algorithms also attempt to minimize performance loss due to such conflicts.

5.1. Scheduling-Binding Problem and Methodology

The inputs to the scheduling and binding algorithm are an application program represented as a set of data flow graphs and some parameters representing the architecture. A dataflow graph is a directed acyclic graph $G(V, E)$, where each element of the vertex set $V = \{v_0, v_1, \dots, v_n\}$ corresponds to an operation, and each edge $e_{ij} \in E$ represents dependency of v_j on v_i . Edge weight d_{ij} corresponding to e_{ij} is the minimum schedule time difference between v_i and v_j , determined by dataflow or control flow dependencies. Each vertex v_i is also associated with a delay x_i that the operation requires to execute. In this graph, v_n is the sink node that has no outgoing edge and has only incoming edges. There can be several sink nodes in the input graph. We assume that each operation v_i may require up to two operands and may produce a result, as specified by r_i^1 , r_i^2 and w_i , which are defined as follows:

$$r_i^j = \begin{cases} 1 & \text{if } v_i \text{ reads operand } j \text{ where } j = 1 \text{ or } 2, \\ 0 & \text{otherwise.} \end{cases}$$

$$w_i = \begin{cases} 1 & \text{if } v_i \text{ writes result,} \\ 0 & \text{otherwise.} \end{cases}$$

The architecture parameters considered include the number of issue slots N , the number of RF read ports R , and the number of RF write ports W .

The scheduling problem for reduced-port RF architecture is to find the integer labeling of the operations $\varphi : V \rightarrow \mathbb{Z}^+$ satisfying the following constraints:

- (i) Schedule time satisfies the dependency constraints due to all edges in the graph:

$$\varphi(v_i) \geq \varphi(v_j) + d_{ji} \quad \forall i, j \text{ s.t. } e_{ji} \in E. \quad (6)$$

- (ii) Total number of operand reads in a cycle should be less than the number of read ports in the RF:

$$\sum_{i: \varphi(v_i)=j} r_i^1 + r_i^2 \leq R \quad \forall j. \quad (7)$$

- (iii) Total number of result writes in any cycle should be less than the number of write ports in the RF:

$$\sum_{i: \varphi(v_i)+x_i=j} w_i \leq W \quad \forall j. \quad (8)$$

- (iv) Total number of operations scheduled in a cycle should also be less than the issue width of the processor:

$$\sum_{i:\varphi(v_i)=j} i \leq N \quad \forall j. \quad (9)$$

To reduce the impact on performance, we reduce the demand of RF read and write ports by using the operands available in the bypass network. As discussed in Section 4.2, if an operand is available in the bypass paths, it is not necessary to read that operand from the RF. Similarly, if all the uses of a result are taken care of by the bypass paths, we may avoid writing it into the RF. To decide if an RF read or write can be avoided, the schedule information φ is used. In the graph G , each incoming dataflow edge to a vertex corresponds to an RF read, and outgoing dataflow edges from a vertex correspond to an RF write. An RF read for operation v_i , associated with a dataflow edge, $e_{ji} = (v_j, v_i)$, can be avoided if

$$\varphi(v_i) - \varphi(v_j) \leq d_{ji} + depth - 1, \quad (10)$$

where *depth* is the number of pipeline stages between Instruction Decode (ID) and Write-Back (WB) stages, also known as *bypass depth*. If an RF read is avoided, the corresponding value of r_i^1 or r_i^2 is toggled to zero. RF write associated with v_i can be avoided (and w_i set to zero) if the write is not global and all the RF reads corresponding to dataflow edges starting from node v_i follow Condition (10). A write is defined as global write if the value being written is read by other basic blocks.

Notice that the actual value of r_i^1 or r_i^2 is available when v_i is scheduled, as the dependent operation is already scheduled. However, w_i is available when all successors of v_i are scheduled.

5.2. RF Port-Aware Scheduling and Binding

We use list scheduling as the base algorithm. In standard list scheduling, in each step the scheduler examines the ready list and a set of nodes ready to be scheduled and schedules them in order of their priorities considering resource constraints. The priority determines which operation should be scheduled earlier when the number of ready operations is more than the resources available.

Our RF PortAware (RPA) scheduling algorithm accounts for the new resource constraints. The concept of reservation tables is used to maintain the list of resources available in each cycle. Along with the scheduling, our scheduling algorithm performs the operation to FU binding explicitly.

Algorithm 1 shows the pseudo-code of the core RPA scheduling. The input to the algorithm is an acyclic dataflow graph G and the processor model M . M includes the

---

ALGORITHM 1: RPA_sched()

Input: G, M

Output: φ

```

1:  $t = 0$ 
2:  $RT = \text{init\_reservation\_table}(M)$ 
3: while all nodes are not scheduled do
4:    $\text{ready\_list}[] = \text{get\_ready\_list}(G, t)$ 
5:    $\text{priority}[] = \text{get\_priority}(\text{list}, G, M)$ 
6:    $\text{bind\_set} = \text{get\_bind\_set}(\text{ready\_list}, \text{priority}, RT, M, t)$ 
7:    $\varphi[\text{bind\_set}] = t$ 
8:    $t = t + 1$ 
9: end while
```

Fig. 6. Reservation table (Issue, R, and W are the number of issue slots, read ports, and write ports, respectively; Max schedule length is calculated for each compilation region).

number of issue slots (N), bypass depth, number of read and write ports of RF (R and W), RF-FU interconnection network (P), and operation mapping to function unit X . First we initialize cycle time t to zero and the Reservation Table (RT) to available resources, which includes FU, read ports, and write ports (lines 1–2). The reservation table records the availability of resources in any cycle. The structure of the reservation table is shown in Figure 6. It has three parts, RT_{FU} , RT_R , RT_W . $RT_{FU}[f, t]$, $RT_R[r, t]$, and $RT_W[w, t]$ indicate the availability of FU f , read port r , and write port w , respectively, at time t . The main loop finds and schedules operations in the current cycle (lines 3–9). For each cycle, a ready list is generated using the *get_ready_list()* function (line 4). Function *get_priority* computes the priority of each operation in the ready list (line 5). We use maximum distance from sink node as the priority function. The set of operations to be scheduled in the current cycle and their mapping to FUs are determined in the *get_bind_set* function (described in Algorithm 2) taking care of constraints (7), (8), and (9). The operations in the *bind_set* are scheduled at the current cycle (line 7). Note that for the purpose of the algorithm, we are using φ as an array rather than a function, with similar semantics. After scheduling operations in the current cycle, t is incremented and the loop is repeated until all the nodes are scheduled.

5.2.1. FU Binding and RF-FU Interconnections. In the underlying architecture, we assume that an FU may perform different types of operations (in other words, we may have *heterogeneous FUs*). FU binding needs to take care of the type of operations that can be executed on different FUs. Further, in direct interconnection, each RF port is shared with a set of specific FU ports, which implies that only one of these FUs can use this port at one time. Thus, the assignment of an FU to an operation may lead to nonavailability of other FUs due to path conflicts. Thus, FU binding in case of direct interconnection needs to handle operation types as well as path conflicts.

Let X be the set of all types of operations and X_i be the set of types of operations that can be executed by the i^{th} FU. A function $\mathcal{T} : V \rightarrow X$ defines the types of operations associated with different nodes of the graph $G(V, E)$.

A valid binding can be defined as an integer labeling $\psi : V \rightarrow \mathbb{Z}^+$ such that

- (i) Each operation is bound to an FU where it can be executed:

$$\mathcal{T}(v_i) \in X_{\psi(v_i)} \quad \forall i. \quad (11)$$

- (ii) No two operations have the same schedule time as well as the same binding option; that is, an FU cannot be used by two operations at the same time:

$$(\varphi(v_i), \psi(v_i)) \neq (\varphi(v_j), \psi(v_j)) \quad \forall i, j, i \neq j. \quad (12)$$

(iii) No two FUs access the same RF port in the same cycle:

$$\sum_{i:\varphi(v_i)=k} (r_i^1 P_{j\psi(v_i)}^1 + r_i^2 P_{j\psi(v_i)}^2) \leq 1 \quad \forall j, k. \quad (13)$$

$$\sum_{i:\varphi(v_i)+x_i=k} w_i P_{j\psi(v_i)}^w \leq 1 \quad \forall j, k. \quad (14)$$

We need a solution that considers all valid OP-FU mappings in a cycle and choose the one that has the maximum number of operations bound in a cycle. For this, we propose a conflict-graph-based heuristic, in which conflicts of all possible mappings are found, and binding is performed on the basis of least conflict.

A node in the conflict graph is a tuple containing the operation and the FU slot, $\langle v_i, f_l \rangle$ if $T(v_i) \in X_l$. Edges in the conflict graph represent conflicts. There is an edge between nodes $\langle v_{i1}, f_{l1} \rangle$ and $\langle v_{i2}, f_{l2} \rangle$ if:

- (i) both operations are mapped to the same FU, that is, $l1 = l2$; or
- (ii) both FUs are accessing the same read port, that is,
 $\exists j \mid (r_{i1}^1 P_{jl1}^1 \text{ or } r_{i1}^2 P_{jl1}^2) \text{ and } (r_{i2}^1 P_{jl2}^1 \text{ or } r_{i2}^2 P_{jl2}^2)$; or
- (iii) both operations write to RF using the same RF port and at the same time, that is,
 $\exists j \mid (w_{i1} P_{jl1}^w \text{ and } w_{i2} P_{jl2}^w = 1) \text{ and } (x_{i1} = x_{i2})$.

With the conflict graph, the binding algorithm binds the least conflicting FU to each operation.

The binding algorithm (*get_bind_set* function) is shown in Algorithm 2. The binding conflict graph is built on the basis of resource requirements of the operations and the FUs to which operations can be mapped (line 2). Binding of a ready operation is done in the order of priority (lines 3–12). All the FUs where an operation can be mapped are considered in increasing order of conflict (line 4). This set of FUs is computed from the binding conflict graph. If all resources (RF read and write ports) required to execute v on the selected FU are available in the reservation table, ψ is set (lines 5–6), and the operation is added to *bind_set* (line 7). Availability of the FU, read ports, and write ports is set in the reservation tables based on usage (line 8), that is, $RT_{FU}[f, t] = 1$; $RT_R[j, t] = 1 \mid r_v^1 P_{jf}^1 \text{ or } r_v^2 P_{jf}^2 \quad \forall j$; $RT_W[k, t + x_v] = 1 \mid w_v P_{kf}^w \quad \forall k$.

ALGORITHM 2: *get_bind_set()*

Input: *ready_list*, *priority*, *RT*, *M*, *t*

Output: *bind_set*, ψ

```

1: bind_set =  $\phi$ 
2: confl_graph = build_conflict_graph(ready_list, RT, M)
3: for  $v \in \text{ready\_list}$  in priority order do
4:   for  $f \in \text{FU}$  in increasing order of conflict do
5:     if resource.available( $v, f$ , RT, t) = 1 then
6:        $\psi[v] = f$ 
7:       bind_set.add( $v$ )
8:       update_res_table( $v, f$ , RT, t)
9:       update_conflict_graph( $v$ , confl_graph)
10:    end if
11:  end for
12: end for
```

Fig. 7. Example dataflow graph.

Table II. Type of Operations That Can Be Executed on Each Function Unit

FU 1 (X_1)	INT	-	MEM	-
FU 2 (X_2)	INT	FLOAT	-	-
FU 3 (X_3)	INT	-	MEM	-
FU 4 (X_4)	INT	-	-	BRANCH

In the conflict graph update, the nodes, conflicting nodes, and nodes related to selected operation are removed with their edges to satisfy constraints (12), (13), and (14).

5.2.2. Example. We illustrate our scheduling and binding algorithm with the help of an example Dataflow Graph (DFG) shown in Figure 7. Each node in the graph represents an operation and different node colors/shades are used for different FU types. The edges between the nodes are either data dependency or control dependency edges. For a simplified view of the DFG, memory dependency, output dependency, and antidependency edges are not shown. Each edge is associated with an edge weight that signifies the minimum time interval between the schedule time of two nodes. For nonunit latency FUs, operands are read in the 0^{th} cycle and results are written in the last cycle. The figure also shows operands read that are available from the RF explicitly in circles (labeled as r1, r2, etc.). Operands that are not read from the RF are assumed to be available in bypass paths.

We schedule this graph for three cases:

- (i) Architecture with reduced read ports and complete interconnection.
- (ii) Architecture with reduced read ports and direct interconnection.
- (iii) Architecture with reduced read and write ports and complete interconnection.

All of these three cases use a four-issue-width processor. The types of operations that can be performed by each issue slot are shown in Table II.

Architecture with reduced read ports and complete interconnection. We schedule the DFG for a four-issue VLIW architecture with a four-read- and four-write-port RF and complete interconnection. For complete interconnection, the only conflict is due to heterogeneous function units. Number of edges on an FU-OP node in the conflict graph

Ready List	Set of Resources Available								Conflict Values				Final Binding		
(In prior. order)									FU1	FU2	FU3	FU4			
	OP1	FU1	FU2	FU3	FU4	R1	R2	R3	R4	5	2	5	2	OP1–FU2	
	OP1														
	OP2	FU1		FU3	FU4			R3	R4	4		4	2	OP2–FU4	
	OP3														
	OP4	FU1		FU3				R4		3		3		OP3–FU1	
OP5															
OP6	OP4		FU3									2			

Fig. 8. Operation binding example.

can be computed as the number of other ready operations that can be mapped to that FU. Overall conflict at a node is the number of edges on it.

In the first cycle, six operations are available in the ready list (shown in Figure 8). The figure shows ready operations in priority order, conflict values of various FUs corresponding to the highest-priority operation ready for schedule, and resources available. In the figure, write-port resources are not shown, as they are not constrained in the current example.

The ready operations are bound in the order of their priority (line 3, `get_bind_set()`) to the least conflicting FU. Since three of the ready operations are MEM type, which can be mapped only on FU1 and FU3, the conflict value is high for FU1 and FU3 and low for FU2 and FU4. OP1, being the first ready operation, is bound to the least conflicting FU2, and the resources required for OP1 (FU2, and two read ports) are removed from the reservation table. The conflict graph is also updated after OP1-FU2 binding. In the same way, OP2 is bound to FU4, and OP3, being a memory operation, is mapped to FU1. None of OP4, OP5, or OP6 could be scheduled in the first cycle due to unavailability of read-port resources.

In cycle 2, due to availability of operands from the bypass paths, OP7, OP8, and OP9 do not require any register file read and therefore, OP7, OP8, OP9, and OP4 are scheduled in this cycle. Similarly, in the third cycle, the available operations are OP10, OP11, OP5, and OP6. Due to availability of operands in the bypass network, the number of read ports required is four instead of seven, and all of these four operations can be scheduled in this cycle. In the last cycle, the remaining three operations are scheduled. The resulting schedule is shown in Figure 9. We observe that in the example DFG, a 50% reduction in RF read ports did not lead to performance degradation due to availability of operands in the bypass network.

Architecture with reduced read ports and direct interconnection. We consider a four-issue-width processor with four read and four write ports. The direct interconnect is as shown in Figure 5(a). Consider the operations of cycle 3 in the scheduled graph shown in Figure 9.

The conflict graph corresponding to the possible mappings is shown in Figure 10. Solid edges in Figure 10(a) show the conflicts due to path conflict and dotted edges in the Figure 10(b) show conflicts due to FUs. We observe in the scheduled graph in Figure 9 that OP10 and OP5 require only the left operand from the RF. OP11 gets both its operands from bypass paths, and OP6 requires both its operands from the RF. The conflict graph is constructed based on this resource requirement and resource information given by interconnection matrix (Figure 5(a)) and type of operations that can be performed by each FU (Table II). Edges in Figure 10(a) are labeled with the

Fig. 9. Schedule for the four-issue slot VLIW processor with four-read-port and four-write-port RF.

(a) Bind graph due to RF port sharing.

(b) Bind graph due to heterogeneous FUs.

Fig. 10. Example bind-conflict graph for cycle 3 of scheduled graph in Figure 9.

RF port causing path conflict. The overall binding conflict graph is formed by the superposition of Figures 10(a) and 10(b) and overall conflict at a node is the sum of all its edges.

The conflict graph operations are bound to FUs on the basis of minimum conflict. To make the example interesting, we assume the order of priority is OP10, OP11, OP5, and OP6, where OP10 is the highest-priority operation. For OP10, FU2 and FU4 are the least-conflicting function units, and we choose FU2 as it is the first available FU. After this binding, the OP10-FU2 node is pruned from the graph along with all

Fig. 11. Example bind-conflict graph for cycle 3 of scheduled graph in Figure 9 after binding of OP5 and OP6.

Fig. 12. Schedule for the four-issue-slot VLIW processor with four-read-port and three-write-port RF.

conflicting mappings, nodes related to OP10, and edges connected to the node. The resulting binding conflict graph is shown in Figure 11(a). Edges due to path conflicts and FU conflicts are drawn in the same graph.

The next operation in order of priority is OP11. Figure 11(a) shows that for OP11, FU1 and FU4 have an equal number of conflicts. FU1 is selected as it is the first available FU and the graph is pruned. The resulting graph is shown in Figure 11(b). Next, OP5 is bound to FU3 and OP6 to FU4 without any conflict.

In contrast to the aforementioned approach, consider a greedy approach where we select the first available FU for each operation under consideration, without considering conflict information. In that case, OP10 would be bound to FU1, OP11 to FU2, and OP5 to FU3, and OP6 would not be bound.

Architecture with reduced read and write ports, and complete interconnection. In this case, we schedule the subject graph for a four-issue-width processor, with four-read-port and three-write-port RF. The resulting schedule (shown in Figure 12) requires five cycles instead of four. The scheduler conservatively assumes that no RF write can be

avoided, so it reserves a resource for all RF writes. Consequently, in all the schedule cycles, a maximum of three RF writes are scheduled.

5.3. Schedule Improvement by Exploiting Write Avoidance

The RPA scheduling described previously is aware of all the constraints of read/write port and interconnection. The write-port resources of an operation are reserved in the current cycle as write avoidance can only be determined in future cycles. Therefore, the resulting schedule does not benefit from the fact that the writes could be avoided due to operand reads from the bypass network.

We reschedule the output of the RPA_sched algorithm, as the information about resources available due to avoided writes is known only after scheduling. We list the operations that can be scheduled in earlier cycles due to availability of write ports. An operation is scheduled if all the resources of the operation are available and other constraints are satisfied. For example, in Figure 12, in the second clock step, resources are available to schedule OP4. With the vacancy created by OP4 in the third cycle, OP5 and OP6 are rescheduled to cycle 3. The new schedule is observed to be the same as the schedule of Figure 9. In this case, we arrive at the optimum schedule in a single iteration. In general, the procedure of improvement can be repeated until we see no further improvement.

Algorithm 3 describes the Im-RPA scheduling strategy that performs iterative schedule improvement. First, the reservation table of each resource is initialized in accordance with the input-scheduled graph (line 1). All the operands for which write can be avoided are determined and their respective write-port resources are freed from the reservation tables (line 5).

ALGORITHM 3: Im-RPA_sched()

Input: V, M

Output: V

```

1: RT = init_reservation_tables(V, M)
2: repeat
3:   for each  $v \in V$  do
4:     remove_available_write_port(RT, v)
5:   end for
6:   for  $t=0$  to schedule length do
7:     moveup_list = find_ready_moveup_op(V, t)
8:     for each  $i \in \text{moveup\_list}$  do
9:       fu_pos = get_fu_position(i, V, M, t)
10:      if fu_pos > 0 then
11:         $\phi[i] = t$ 
12:         $\psi[i] = \text{fu\_pos}$ 
13:        update_res_table(i, fu_pos, RT, t)
14:      end if
15:    end for
16:  end for
17: until schedule length reduction > 0

```

For each schedule cycle, *moveup_list* is calculated (line 8). *moveup_list* is the list of those operations that can be scheduled in the current cycle. All the operations in the *moveup_list* are checked for resource constraints. If the required resources for an operation are available, that operation is scheduled and bound, and the reservation table is updated (lines 10–14). By the end of the loop (lines 7–17), we have a new

schedule. If the new schedule has a smaller schedule length, the whole process is repeated until we benefit in terms of reduction in schedule length.

5.3.1. Algorithm Complexity. The maximum number of iterations in Algorithm 3 is less than the initial schedule length achieved by *RPA_sched*. This can be seen as follows. In the first iteration, possible changes in the 0^{th} cycle of schedule are finalized since all possible *move_up* operations have been considered for cycle 0. Similarly, after the decisions taken in the k^{th} cycle, the schedule for cycles 0– k will not change. Thus, the maximum number of iterations would always be less than the initial schedule length of the graph.

6. EXPERIMENTS AND RESULTS

6.1. Experimental Setup

For our experiments, we have used the Trimaran compiler framework [Chakrapani et al. 2005] after enhancing the machine description and modifying the scheduling algorithm. The Trimaran compiler framework is an open-source retargetable compiler-simulator infrastructure for the VLIW class of processors. Trimaran takes as input the number of FUs, type of FUs, size of register file, and instruction set. We augmented the machine description with additional information about the FU-RF interconnection and RF port information. The scheduling and binding algorithms are implemented in the back-end compiler of Trimaran known as *Elcor*, after performing all instruction-level optimizations. The process of code generation and simulation is fully automated with no manual intervention. To avoid the effect of number of registers in register files (e.g., spill code), we simulated the scheduled code with virtual registers. The effect of virtual registers is discussed in the next section.

We use a processor with a fixed issue width with all possible bypass paths available. The issue width used in commercial VLIW processors varies from two to 16; however processors with issue widths of four are most common [Faraboschi et al. 2000; Seshan 1998]. Therefore, we use a four-issue VLIW processor as the base processor with four integer units, one floating point unit, two memory units, one branch unit, and a 64-word register file. Function units are placed in different issue slots, as shown in Table II. Experiments were performed with a varying number of RF read and write ports. In the experiments, we refer to an RF configuration by the number of read and write ports; for example, a 4r3w configuration represents a shared-port RF with four read ports and three write ports. Each RF configuration can have a direct or complete interconnection except 8r4w, where only direct interconnection is present.

We have used CACTI 4.2 [Tarjan et al. 2006] to estimate read and write access energy for the direct interconnect RF. For complete interconnect RF, we modeled interconnects in CACTI. We assumed 70nm technology, 85°C temperature [Skadron et al. 2004], 1GHz frequency, 64-bit 64-word RF for calculating the energy.

We evaluated our proposed approach with two sets of benchmarks. Set I benchmarks are composed of Mediabench [Lee et al. 1997] and Mibench [Guthaus et al. 2001]. Mediabench and Mibench both consist of embedded applications. The source code of Set I benchmarks is used as is, and not modified to increase ILP to maintain their original characteristics. Set II benchmarks consist of a number of kernels and applications from an embedded systems domain with high InstructionLevel Parallelism (ILP). Certain transformations such as loop unrolling [Davidson and Jinturkar 1995], constant folding, and tree height reduction [Mahlke et al. 1992] are used to further enhance the ILP of Set II benchmark applications.

The two sets of benchmarks have different resource requirements such as RF read, RF writes, and FU required in a cycle. Therefore, the two sets are suitable to evaluate shared-port RF architecture. Resource requirement can be characterized by the ILP

Table III. Benchmark Characteristics

Set I		Set II	
Benchmarks	ILP	Benchmarks	ILP
basicmath	1.12	mm_int32	18.29
dijkstra	1.03	dct_int2	13.4
bitcount	1.82	sobel	10.96
blowfish	2.38	convolution	17.48
FFT	1.31	hamm	28.96
patricia	1.13	colorspace	16.76
qsort	1.26	mm_int8	4.63
sha	3.03	dct_int	7.44
g721encode	1.36	susan	4.29
g721decode	1.37	viterbi	4.78
gsmdecode	1.39	rijndael	4.4
gsmencode	2.23		
unepic	1.23		
rawcaudio	1.44		
rawaudio	1.52		
pegwitdec	1.46		
pegwitenc	1.42		

in the application. Notice that the intrinsic ILP in an application is independent of the processor and issue width, though the achievable ILP depends on the compiler. We compute ILP in an application by compiling and simulating the benchmarks for a very wide issue processor so that resources are not the constraint. We use a 64-issue processor for this purpose. The ILP value for each benchmark is shown in Table III.

Our overall objective is to achieve reduction in the RF energy with very little performance penalty. We first present performance results for the benchmarks, followed by energy results.

6.2. Number of Cycles

6.2.1. Direct Interconnect Evaluation. We evaluated different interconnection matrices to understand the gains due to the guidelines discussed in Section 4.1.1. We have used the 4r4w configuration for this experiment, which has maximum write ports and reduced read ports. We used a sample of 24 different interconnection matrices (out of 652) and grouped them by their RF port imbalance factor. All the configurations having “0” RF port imbalance form one group, configurations having “1” RF port imbalance form a separate group, and so on.

The average percentage increase in the number of cycles for each group for all benchmarks (of Set I) with respect to the 8r4w configuration is shown in Figure 13. We clearly observe that the higher the imbalance, the higher the performance penalty. In other words, if an interconnect matrix follows all the three guidelines, the performance penalty would be the least and may come close to that of a complete interconnection.

With this insight, the interconnection matrices for other RF configurations are selected based on the minimum port imbalance and are shown in Table IV. The table shows four entries for each RF port configuration. These four entries are for four different issue slots. Each entry contains three digits showing the two read and one write port of RF to which that issue slot is connected. Some of these configurations are shown in Figure 14.

6.2.2. Model Validation and Performance Evaluation of the Proposed Architecture. Our analytical model (Section 3.2) approximates the execution time of an application

Fig. 13. Direct interconnection RF architecture exploration.

Table IV. Interconnect Matrices for Different Direct RF Configurations

	8r4w	8r3w	8r2w	8r1w	6r4w	6r3w	6r2w	6r1w
Slot 1	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
Slot 2	2 3 1	2 3 1	2 3 1	2 3 0	2 3 1	2 3 1	2 3 1	2 3 0
Slot 3	4 5 2	4 5 2	4 5 0	4 5 0	4 5 2	4 5 2	4 5 0	4 5 0
Slot 4	6 7 3	6 7 2	6 7 1	6 7 0	5 4 3	5 4 2	5 4 1	5 4 0

	4r4w	4r3w	4r2w	4r1w	2r4w	2r3w	2r2w	2r1w
Slot 1	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
Slot 2	1 2 1	1 2 1	1 2 1	1 2 0	1 0 1	1 0 1	1 0 1	1 0 0
Slot 3	2 3 2	2 3 2	2 3 0	2 3 0	0 1 2	0 1 2	0 1 0	0 1 0
Slot 4	3 0 3	3 0 2	3 0 1	3 0 0	1 0 3	1 0 2	1 0 1	1 0 0

by considering the application characteristics, such as parallelism and resource requirement. First, we validate the model by code generation and simulation for the corresponding architectures.

We compiled and simulated the benchmark applications for different register file configurations and normalized the execution cycles with respect to cycles of the 8r4w configuration. Normalized cycles for all the benchmarks are averaged. Figure 15 shows the average normalized cycles for different RF configurations as estimated by the performance model (Section 3) and as obtained by the code generation and simulation of individual RF configurations.

Figure 15(a) shows the performance comparison for Set I benchmark applications. We observe that the performance estimate of our model is always within 2% of the performance obtained by simulations. For Set II benchmarks (Figure 15(b)), the performance difference is within 12%. These results clarify that the performance model closely models the behavior of the RTA scheduling algorithm, and thus, can be used for design space exploration.

In the experiment described previously, the performance predicted by the model is compared with the performance of *complete interconnect* architecture, since our performance model assumes that there are no path conflicts. However, the model may be used for estimating the performance of direct interconnects also, as the performance difference between complete interconnection and direct interconnection architecture using our RPA scheduler is marginal. To demonstrate this, Figure 16 shows the average

Fig. 14. Different direct RF configurations.

normalized cycles for different RF configurations for direct interconnect and complete interconnect.

Figures 16(a) and 16(b) show the average normalized number of cycles for some selected configurations with *direct interconnect* and *complete interconnect* architectures. Both figures have three pairs of curves. The first pair represents a decreasing number of write ports; the second represents a decreasing number of read ports; and the third represents decreasing values of both read and write ports. It is clear that complete interconnect always performs better than the direct interconnect, because of the absence of path conflicts. For Set I benchmarks, the average number of cycles over all benchmarks in the architecture with direct interconnect is within 2% of the number of cycles in the architecture with complete interconnect, for any RF configuration. In case of Set II benchmarks, this figure is 8%.

The first graph in Figure 16(a) and 16(b) shows the variation of average normalized cycles with decreasing write ports. The number of cycles increases by 60% for single-write port, but it is only 9% and 20% for three-write- and two-write-port cases, respectively, in case of Set II benchmarks. This shows that performance deteriorates rapidly with decreasing write ports. On the other hand, the effect of reducing read ports on the performance is less drastic. The second set of graphs in the same figure shows the performance variation with the read ports keeping write ports fixed. There is almost no difference between the performance of six-read-port and eight-read-port configurations. In case of a four-read-port configuration, the performance loss is marginal at 5% for Set II benchmarks and no performance loss for Set I benchmarks. The increase in the number of cycles is significant only when read ports are reduced to two.

(a) Normalized average number of cycles for set I.

(b) Normalized average number of cycles for set II.

Fig. 15. Model validation against simulation results.

The third pair of graphs shows the impact on the number of cycles when both read and write ports decrease in the same ratio. For these configurations, the increase in the number of cycles is both due to read port reduction as well as due to write port reduction, as estimated by Equation (4); that is, the overall increase in the number of cycles for a configuration is approximately equal to the sum of the individual increases in cycle count due to read-port constraints and write-port constraints.

There is a marked difference between the performance observed for Set I benchmarks (Figure 16(a)) and Set II benchmarks (Figure 16(b)). The performance penalty due to

(a) Normalized average number of cycles of selected configurations for set I.

(b) Normalized average number of cycles of selected configurations for set II.

Fig. 16. Performance evaluation.

port and path conflicts for Set II is much larger than that for Set I benchmarks because of higher demand of read and write ports in the applications.

Effect of limited number of registers. As mentioned in Section 6.1, simulations were performed with virtual registers; that is, register allocation was not performed during these experiments. By using virtual registers, we ignored the effect of spill code due to limited registers and the effect of shared register address space with pipeline registers.

There are additional loads and stores that need to be scheduled due to spill code. These additional operations may stretch the schedule length. In typical register allocation algorithms, virtual registers with the largest distance between definition and use are stored in memory. In unlimited register architectures, these registers are always read from the register file. When these registers are stored in memory (as spill code), the distance between definition and use decreases and there is a high probability that the register operand for spilled-store is read from the bypass path. Similarly, the result of spill-load operation is available in bypass paths for its use. Further, since spill load and store use stack address space, address operand of load/store instruction is an immediate operand, and thus, may not require an RF port. If the operands of spill-store are available in the bypass path and the result of spill-load is used in bypass paths, then these load/stores would not require any RF port for the data operand. Due to a reduced resource requirement, there is a positive impact on the performance.

Shared register address space with pipeline registers also reduces the availability of physical registers and may lead to more spill code. However, it has been shown that pipeline register-aware register allocation [Yan and Zhang 2008] is very effective in reducing the spills.

6.3. Energy

In conventional processors, bypass paths are present to avoid performance loss due to data dependencies, but RF access is not avoided even if values are found in bypass paths. We compare RF energy in our proposed architecture with RF energy consumed in conventional processors.

For RF energy computation, we use Equation (1) and the processor configuration given in Section 6.1. In Section 3, we assumed that the number of reads and writes do not change with the number of RF read and write ports. Characteristic vectors $\mathbf{R}$ and $\mathbf{W}$ give the number of reads and writes, respectively, in case of unconstrained (eight read, four write port) RF. Based on this, average normalized RF energy is computed for different RF configurations and is plotted in Figure 17 with “*.” The values are normalized considering the RF energy consumed in conventional processors as the base.

This graph shows the theoretical advantage of a reduced-port register file. In case of eight-read four-write RF, the energy reduction is due to avoided reads and writes, while in other configurations, the energy reduction is also due to reduced RF access energy. We notice that energy savings vary from 17% (in case of RF without port reduction) to 75% for two-read- and one-write-port cases for both sets of benchmarks. Note that this energy computation does not take into account the effect of RF port reduction on the number of reads/writes and additional execution cycles.

With our simulation infrastructure, we obtained the number of reads and writes for every application and every RF configuration. Using these numbers, we computed RF energy for direct interconnection and plotted with “x” in Figure 17. The RPA scheduling algorithm inherently increases the probability of the reads and writes from bypass paths in case of port reduction, and thus decreases the number of RF reads and writes, as is evident in the experiments. However, when the port reduction is high (e.g., in configurations with one write port), the instruction schedule stretches considerably. Due to this, we miss several reads and writes from bypass paths and leakage energy contribution is larger. Because of high parallelism, these effects are more evident in Set II benchmarks.

In complete interconnect, one input is selected using a multiplexer from all inputs received at each FU input port. We model energy consumed by these multiplexers using the NAND gate model provided in CACTI. With the modified RF access energy, we computed the RF energy for complete interconnect, shown with “+” in Figure 17. We

(a) Normalized average RF energy for benchmarks in Set I.

(b) Normalized average RF energy for benchmarks in Set II.

Fig. 17. Normalized average RF energy for the direct and the complete interconnect topologies.

observe that the energy gains by port reduction have been partially offset by additional energy consumed in the RF-FU interconnection.

In summary, reduced-port RF saves energy due to a number of factors: first, due to avoidance of RF access; second, due to reduced RF access energy; and third, due to a decrease in RF accesses due to the RPA scheduling algorithm. Furthermore, there is no energy overhead of the proposed architecture in case of direct interconnection, as discussed in Section 4.

Figure 18 shows the normalized RF energy of the 8r4w configuration and the 4r4w configuration with complete interconnection and direct interconnection for different benchmarks. The number of reads from bypass paths is similar in both direct interconnect and complete interconnect. The energy saving in the 8r4w configuration is due to fewer reads and writes. Therefore, “rijndael,” with the fewest reads/writes from the RF, saves the most energy. For some benchmarks, such as “gsmdecode,” energy saving in

(a) Normalized RF Energy for Set I benchmarks.

(b) Normalized RF Energy for Set II benchmarks.

Fig. 18. Normalized RF energy for different benchmarks in 8r4w and 4r4w configurations.

the 4r4w configuration with respect to the 8r4w is high, while in “gsmencode,” it is low. The reduction in the energy due to RF dynamic access energy is similar; the difference is due to leakage energy, which depends on the number of cycles.

6.3.1. Processor Energy. Increase in runtime implies that contribution to the energy consumption due to leakage current increases. However, it does not offset the reduction in energy resulting from port reduction and RF access reduction. For illustration, consider the energy savings and performance loss by a typical proposed RF configuration. A typical proposed RF configuration saves 60% of energy with a performance loss less than 5% (on Set I benchmarks). Further, consider that 20% energy is consumed by RF. Energy of the rest of the processor may increase almost linearly with an increase in number of cycles. Thus, the net energy savings in this case can be $20 \times 0.60 - 80 \times 0.05$, that is, 8% of total CPU energy.

For ILP benchmarks (benchmark Set II in our experiments), where the runtime increase due to port reduction is more pronounced, the overall reduction in energy is smaller.

(a) Normalized average RF energy-delay product for benchmarks in Set I.

(b) Normalized average RF energy-delay product for benchmarks in Set II.

Fig. 19. Normalized average RF energy-delay product for the direct and the complete interconnect topologies.

6.4. Finding a Good Configuration

In the previous sections, we observed a number of tradeoffs. The *complete interconnection* architecture shows good performance, but it is not as energy efficient as the direct interconnection architecture. Similarly, architecture with fewer ports saves more energy, but the associated performance penalty is also more. Thus, choosing only energy or performance as criteria may not lead to a good architecture.

There can be several other metrics for choosing an optimal configuration, such as minimum acceptable performance loss or energy budget for the system, energy delay product, or a combination of these.

We computed the product of RF energy and number of cycles and normalized it against conventional RF with no port reduction. The average and normalized energy delay product is plotted in Figure 19. According to this criterion, two-read- and

one-write-port RF configuration with direct interconnect is the best configuration for Set I benchmarks. This configuration saves 83% energy with a performance overhead of 10%. For Set II benchmarks, four-read- and four-write-port RF with direct interconnect is the best configuration. This configuration saves 47% of RF energy with a performance penalty of 6%.

In the best case, assuming 20% energy is consumed in the RF, $20 \times 0.83 - 80 \times 0.1 = 8.6\%$ of total CPU energy can be saved in case of Set I benchmarks and 4.6% of total CPU energy is saved in case of Set II benchmarks.

7. CONCLUSIONS

In this article, we proposed a shared-port RF architecture to reduce the energy of VLIW processors. Different interconnection topologies were studied. Our study shows that complete interconnection is less energy efficient than direct interconnection. Though the direct interconnection leads to more compiler constraints, the performance losses were found to be within 2% of the complete interconnection topology for the set of Mediabench and Mibench benchmarks. We showed that compiler support is important for the architecture. We proposed algorithms for scheduling and binding, which use bypass awareness to reduce read and write demand and the minimum-conflict-based method for operation to FU binding. With bigger reduction of ports, higher performance penalties were observed, though these configurations offer higher register file energy savings. The best shared-port RF configuration leads to 83% energy savings in the RF energy for Mediabench and Mibench benchmarks with a performance penalty of only 10%. Finally, we observe that the number of ports in the RF can be selected on the basis of energy budget or performance budget.

In the future, we plan to study the effects of register allocation on performance of shared-port RF architecture.

REFERENCES

- S. Aditya, B. R. Rau, and V. Kathail. 1999. Automatic architectural synthesis of VLIW and EPIC processors. In *International Symposium on System Synthesis*. 107–113.
- R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi. 2001. Reducing the complexity of the register file in dynamic superscalar processors. In *Proceedings. 34th Annual International Symposium on Microarchitecture*. 237–248.
- A. Capitanio, N. Dutt, and A. Nicolau. 1992. Partitioned register files for VLIWs: A preliminary analysis of tradeoffs. In *Proceedings 25th Annual International Symposium on Microarchitecture*.
- L. N. Chakrapani, J. Gyllenhaal, W. M. W. Hwu, S. A. Mahlke, K. V. Palem, and R. M. Rabbah. 2005. Trimaran: An infrastructure for research in instruction-level parallelism. In *Lecture Notes in Computer Science*. Vol. 3602/2005. Springer, Berlin, 32–41.
- H. Corporaal. 1999. TTAs: Missing the ILP complexity wall. *Journal of Systems Architecture* 36, 12 (1999).
- J. L. Cruz, A. González, M. Valero, and N. P. Topham. 2000. Multiple-banked register file architectures. In *International Symposium on Computer Architecture*. 316–325.
- J. W. Davidson and S. Jinturkar. 1995. Improving instruction-level parallelism by loop unrolling and dynamic memory disambiguation. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*.
- J. R. Ellis. 1986. *Bulldog: A Compiler for VLSI Architectures*. MIT Press, Cambridge, MA.
- O. Ergin, D. Balkan, K. Ghose, and D. Ponomarev. 2004. Register packing: Exploiting narrow-width operands for reducing register file pressure. In *Proceedings of the 37th Annual International Symposium on Microarchitecture*. 304–315.
- K. Fan, N. Clark, M. Chu, K. V. Manjunath, R. Ravindran, M. Smelyanskiy, and S. Mahlke. 2003. Systematic register bypass customization for application-specific processors. In *Proceedings of the IEEE 14th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*.
- P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood. 2000. Lx: A technology platform for customizable VLIW embedded processing. In *Proceeding of the 27th International Symposium on Computer Architecture*. 203–213.

- N. Goel, A. Kumar, and P. R. Panda. 2007. Power reduction in VLIW processor with compiler driven bypass network. In *Proceedings of the International Conference on VLSI Design and Embedded Systems*. 233–238.
- R. L. Graham, D. E. Knuth, and O. Patashnik. 1988. *Concrete Mathematics*. Addison-Wesley.
- M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*.
- J. Hoogerbrugge and H. Corporaal. 1994. Register file port requirements of transport triggered architectures. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*. 191–195.
- V. Kathail, M. Schlansker, and B. R. Rau. 2000. *HPL-PD Architecture Specification: Version 1.1*. Technical Report HPL-93-80R1.
- N. S. Kim and T. Mudge. 2003. Reducing register ports using delayed write-back queues and operand pre-fetch. In *Proceedings of the International Conference on Supercomputing*. 172–182.
- M. Kudlur, K. Fan, M. Chu, R. Ravindran, N. Clark, and S. Mahlke. 2004. FLASH: Foresighted latency-aware scheduling heuristic for processors with customized datapaths. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO'04)*. 201–212.
- A. Lambrechts, P. Raghavan, A. Leroy, G. Talavera, T. V. Aa, M. Jayapala, F. Catthoor, D. Verkest, G. Deconinck, H. Corporaal, F. Robert, and J. Carrabina. 2005. Power breakdown analysis for a heterogeneous NoC platform running a video application. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architecture Processors*. 179–184.
- C. Lee, M. Potkonjak, and W. H. Mangione-Smith. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the International Symposium on Microarchitecture*. 330–335.
- P. G. Lowney, S. M. Freudenberger, T. J. Karzes, W. D. Lichtenstein, R. P. Nix, J. S. O'Donnell, and John Ruttenberg. 1993. The multiflow trace scheduling compiler. *Journal of Supercomputing* 7 (May 1993), 51–142.
- S. A. Mahlke, W. Y. Chen, J. C. Gyllenhaal, W. W. Hwu, P. P. Chang, and T. Kiyohara. 1992. Compiler code transformations for superscalar-based high-performance systems. In *Proceedings of Supercomputing '92*. 808–817.
- C. H. Papadimitriou and K. Steiglitz. 1998. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications.
- I. Park, M. D. Powell, and T. N. Vijaykumar. 2002. Reducing register ports for higher speed and lower energy. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*. 171–182.
- S. Park, A. Shrivastava, N. Dutt, A. Nicolau, Y. Paek, and E. Earlie. 2006. Bypass aware instruction scheduling for register file power reduction. In *Proceedings of the Conference on Language, Compilers, and Tool Support for Embedded Systems*. 173–181.
- S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. K. Kapasi, and J. D. Owens. 2000. Register organization for media processing. In *Proceedings of the International Symposium on High Performance Computer Architecture*. 375–386.
- M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon. 2002. Low-power data forwarding for VLIW embedded architectures. *IEEE Transaction of VLSI Systems* 10, 5 (2002), 614–622.
- R. Sangireddy. 2007. Register port complexity reduction in wide-issue processors with selective instruction execution. *Microprocessors and Microsystems* 31, 1 (2007), 51–62.
- N. Seshan. 1998. High Velocity processing. *IEEE Signal Processing Magazine* 15, 2 (1998), 88–101.
- P. Shivakumar and N. P. Jouppi. 2001. *Cacti 3.0: An Integrated Cache Timing, Power and Area Model*. Technical Report 2001/2. Western Research Laboratory.
- A. Shrivastava, N. Dutt, A. Nicolau, and E. Earlie. 2005. PBExplore: A framework for compiler-in-the-loop exploration of partial bypassing in embedded processors. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'05)*. 1264–1269.
- S. Sirsi and A. Aggarwal. 2009. Exploring the limits of port reduction in centralized register files. In *Proceedings of the 22nd International Conference on VLSI Design and Embedded System*. 535–540.
- K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. 2004. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.* 1, 1 (2004), 94–125.
- D. Tarjan, S. Thoziyoor, and N. P. Jouppi. 2006. *CACTI 4.0*. Technical Report HPL-2006-86, HP Labs. (2006).
- J. W. van de Waerd, S. Vassiliadis, S. Das, S. Mirolo, C. Yen, B. Zhong, C. Basto, J. P. van Itegem, D. Amirharaj, K. Kalra, P. Rodriguez, and H. van Antwerpen. 2005. The TM3270 media-processor. In *Proceedings of the International Conference on Microarchitecture*. 331–342.

- J. T. J. van Eijndhoven, F. W. Sijstermans, K. A. Vissers, E. J. D. Pol, M. I. A. Tromp, P. Struik, R. H. J. Bloks, P. van der Wolf, A. D. Pimentel, and H. P. E. Vranken. 1999. TriMedia CPU64 architecture. In *Proceedings of the International Conference on Computer Design*. 586–592.
- J. Yan and W. Zhang. 2008. Exploiting virtual registers to reduce pressure on real registers. *ACM Trans. Archit. Code Optim.* 4, 4 (2008), 1–18.
- J. Zalamea, J. Llosa, E. Ayguade, and M. Valero. 2000. Two-level hierarchical register file organization for VLIW processors. In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*. 137–146.
- V. Zyuban and P. Kogge. 1998. The energy complexity of register files. In *International Symposium on Low Power Electronics and Design*. 305–310.

Received March 2011; revised August 2013; accepted September 2013