Compiler-Assisted Energy optimization for Clustered VLIW Processors

Rahul Nagpal, Y. N. Srikant[1]

Dept. of CSA, Indian Institute of Science, Bangalore, KA, India

Abstract

Clustered architecture processors are preferred for embedded systems because centralized register file architectures scale poorly in terms of clock rate, chip area, and power consumption. Although clustering helps by improving clock speed, reducing energy consumption of the logic, and making the design simpler, it introduces extra overheads by way of inter-cluster communication. This communication happens over long global wires having high load capacitance which leads to delay in execution and significantly high energy consumption. Inter-cluster communication also introduces many short idle cycles, thereby significantly increasing the overall leakage energy consumption in the functional units. The trend towards miniaturization of devices (and associated reduction in threshold voltage) makes energy consumption in interconnects and functional units even worse and limit the usability of clustered architectures in smaller technologies. However, technological advancements now permit design of interconnects and functional units with varying performance and power modes. In this paper, we propose scheduling algorithms that aggregate the scheduling slack of instructions and communication slack of data values to exploit the low power modes of functional units and interconnects. Finally, we present a synergistic combination of these algorithms that simultaneously save energy in functional units and interconnects to improve the usability of clustered architectures by achieving better overall energy-performance trade-offs. Even with the conservative estimates of contribution of functional unit and interconnect to overall processor energy consumption, the proposed combined scheme obtains on an average 8% and 10% improvement in overall energy-delay

Email addresses: rahul@csa.iisc.ernet.in (Rahul Nagpal), srikant@csa.iisc.ernet.in (Y. N. Srikant)

[1]Corresponding Author. Address: Dept of CSA, Indian Institute of Science, Banagalore, Karnataka, India 560012

product with 3.5% and 2% performance degradation for a 2-clustered and a 4-clustered machine respectively. We present a detailed experimental evaluation of the proposed schemes. Our test bed uses the Trimaran compiler infrastructure.

Keywords: Scheduling, Clustered VLIW Processors, Energy-Aware Scheduling

1. Introduction

Proliferation of embedded systems has opened up many new research issues. Design challenges posed by embedded processors are ostensibly different from those offered by general purpose systems. Apart from very high performance they also demand low power consumption, low cost, and less chip area to be practical. The ever increasing trend towards miniaturization of devices makes utilizing huge transistor budget in a manner that enables high clock speed, low design complexity, and less energy consumption(1) even more challenging. However, resolving this challenge can enable the deployment of embedded systems for many performance-demanding never-before embedded applications at a lower cost. Another challenge posed by this technological advancement is the rising level of the leakage energy consumption in the logic. The increase in the transistor density requires reducing the supply voltage in order to operate the circuit reliably. The reduction in supply voltage also requires reduction in the threshold voltage in order to maintain the speedup and this leads to an exponential rise in the leakage component of the energy consumption(2). With the 65nm and smaller technologies currently in fabrication, the leakage energy is on par with the dynamic energy consumption. In future technologies the leakage energy will further dominate the overall energy consumption(3).

Distribution or clustering is the common design theme that is being employed in one form or another to meet these challenges. The basic idea is to design simpler and smaller components and put together a collection of these components interconnected using a communication fabric. Smaller components are simpler to design, enable faster clock speed, and incur less energy consumption. Different architectural philosophies(4)(5)(6)(7)(8)(9) have used distribution in its varied form to tackle the scalability problem in the past. This trend is expected to continue in the future also with ever growing number of transistors on the chip.

Clustered VLIW architectures(10)(11)(4) use clustering philosophy in context of VLIW architectures. These architectures are being widely adopted in embedded domain because they overcome the scalability problem associated with centralized VLIW architectures. A clustered VLIW architecture(4) has more than

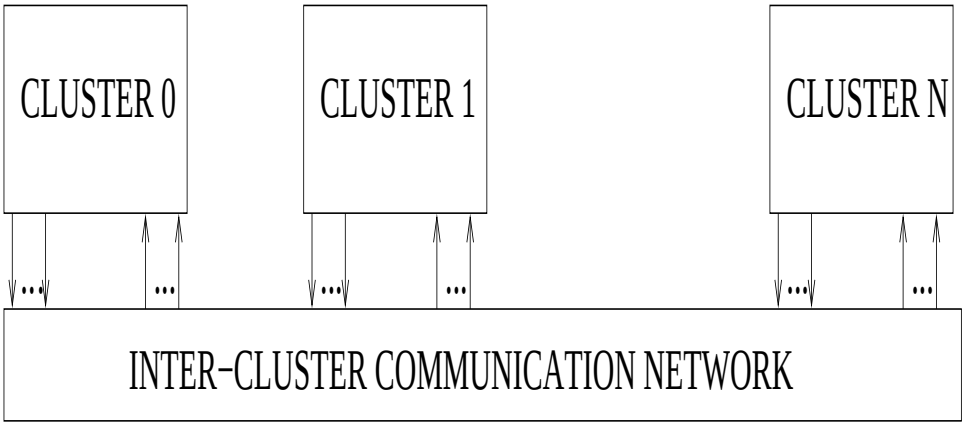
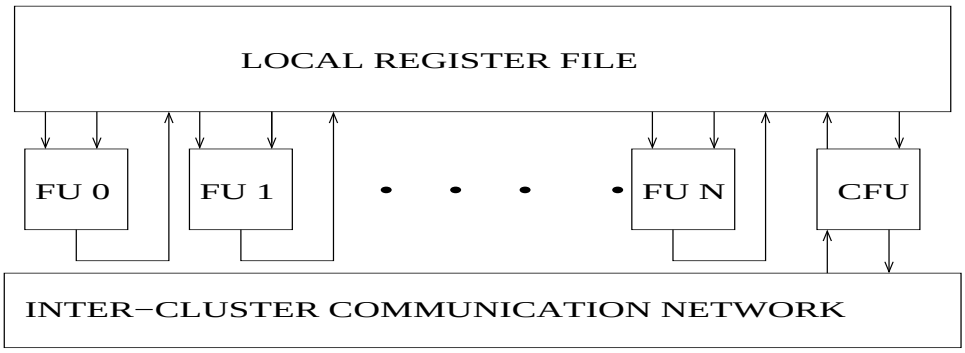


Figure 1: A General Clustered VLIW Architecture



An Individual Cluster

FU Function Unit
CFU Communication Function Unit

Figure 2: A Cluster

one register file and connects only a subset of functional units to a register file (see Figure 1 and 2). Groups of small computation clusters can be interconnected using some interconnection topology and communication can be enabled using any of the various inter-cluster communication models(12). Clustering avoids area and power consumption problems of centralized register file architectures while retaining high clock speed which can be leveraged to get better performance. Texas Instrument's VelociTI(13), HP/ST's Lx(14), Analog's TigerSHARC(15), and BOPS' ManArray(16) are examples of the architectures developed on the basis of clustered ILP philosophy. IBM's eLite(17) is a research proposal for a novel clustered architecture. Clustered VLIW architectures continue to be popular in embedded domain and are part of some of the most popular and recent chips planned to power smart phones and tablets (18) apart from their presence in low-end phones(19).

Though clustering helps to combat the scalability problem by making components simpler and thereby increasing clock rate and reducing dynamic energy consumption of functional components, an interconnection network is required for the communication of data values among different components. This communication in clustered architectures happens over long wires having high load capacitance which in effect takes more time and incurs more energy consumption(1)(20). This problem is becoming severe with each upcoming process technology. As a result, clustered architectures are becoming more communication bound in terms of the performance and energy consumption. Apart from the interconnects, functional units are another major source of energy consumption in clustered architectures. The frequent accesses to functional units raises the temperature level and makes the leakage energy consumption which is specifically a concern in smaller technologies even worse. Moreover, the contention for limited number of slow interconnects leads to many short idle cycles and that further increases the leakage energy consumption in functional units.

Clustered VLIW architectures rely on compile-time scheduling. The static scheduling simplifies the issue logic by alleviating the need for a dedicated hardware for scheduling. Thus, a significant fraction of the total energy consumption in clustered VLIW architectures is attributed to interconnects and functional units. Though, the exact percentage depends upon the architecture and circuit details, earlier studies report that a very high percentage (25% to 30%) of total processor energy consumption is attributed to interconnects. Similarly a large fraction (30% to 35%) of static energy consumption in a VLIW architecture is attributed to functional units(21). An architecture level model developed in (22) also confirms that the leakage energy consumption in functional units constitutes a noticeable frac-

tion of the overall processor leakage energy consumption despite having a smaller transistor count compared to the caches. Thus, optimizing energy in interconnects and functional units in clustered architectures is becoming more and more important from one process generation to another.

However, the functional units and interconnects are often underutilized in clustered VLIW architectures. Apart from other usual causes such as data dependencies, the under-utilization of functional units is also due to the contention for limited number of slow interconnect channels that introduces many short idle cycles for functional units. At the same time since the functional units are distributed among clusters, there is also more contention for functional resources which leads to underutilization of interconnects. Finally, the contention for functional and interconnect resources in clustered VLIW architecture combine in a synergistic fashion and lead to greater available slack in clustered architectures as compared to VLIW architectures.

The advancements in VLSI technology now enable designing interconnects and functional units with different power and performance modes. For example (23)(24) show that using 45nm technology, it is possible to design wires consuming 1/5 the energy but having twice the delay(23). (25) proposes to use interconnect composed of wires with different characteristics to improve the ED^2[1] of the superscalar processor. Similarly the capabilities of dual-threshold domino logic with sleep mode (that can transition between active mode and sleep mode and vice versa without any performance penalty(26) but with moderate energy penalty) can be utilized to do leakage energy management for short idle cycles in functional units. One such purely hardware based scheme in the context of a superscalar architecture is due to Dropsho et al.(27). Their scheme puts any integer ALU into low leakage mode after one cycle of idleness. Their results confirm the benefits of such an aggressive scheme in smaller technologies. However, being a purely hardware based scheme, the benefits are severely (on average, by 30%) affected by frequent transitions from active mode to sleep mode and vice-versa because of many short idle periods.

In this paper, we propose a compiler-directed approach that leverages on these advancements in VLSI technology to improve the usability of clustered VLIW architecture in smaller technologies, targeting the two major source of energy consumption namely interconnects and functional units. Though, there has been some work in the past to reduce leakage energy consumption in functional units in

[1] ED^2 is defined as Energy*Execution Time* Execution Time

the context of superscalar and VLIW architectures, to the best of our knowledge, there has been no such work in the context of clustered VLIW architectures specifically targeting smaller technologies. Regarding interconnects, the primary focus of research had been to reduce the latency of communication. We are not aware of any work that targets to reduce energy consumption in interconnects in clustered VLIW architectures. In context of inter-cluster communication, we limit our focus on most popular inter-cluster communication models(12) such as explicit inter-cluster communication through inter-cluster move instructions and extended operand inter-cluster communication models(12) found in commercial clustered processors such as Texas Instrument's VelociTI(13) and HP/ST's Lx(14). The novelty of our approach also lies in an integrated scheduling algorithm that simultaneously reduces the energy consumption in functional units as well as interconnects. The contention for a limited number of functional and communication resources in a clustered VLIW architecture leads to increased cycles of execution on a clustered machine as compared to an equivalent VLIW machine. Our approach aggregates the scheduling slack of instructions and communication slack of data values in a synergistic fashion to convert the inherent idleness of functional and communication resources in clustered architecture to energy gains. The major contributions of our approach can be stated as follows:

- A scheduling algorithm for clustered-VLIW architectures that exploits the scheduling slacks with an aim of reducing the number of transitions and associated overheads thereby significantly improving the leakage energy consumption compared to the underlying architectural scheme.

- Another scheduling scheme for clustered architectures that exploits communication slack of data values and scheduling slack of instructions to reduce the energy consumption in interconnects while achieving better performance for clustered architectures. The proposed scheme provides performance comparable to a dual bandwidth clustered architectures at nearly half the energy cost.

- An integrated scheme that simultaneously exploits scheduling slack of instructions and communication slack of data values to achieve better overall energy savings. This scheme converts any inherent performance loss due to contention for communication and computation resources into energy benefits.

- We have significantly extended Trimaran Compiler Framework to faithfully

model different clustered VLIW configurations and inter-cluster communication models. We have implemented these schemes in extended Trimaran framework. We present a detailed performance analysis based on experimental evaluation of these algorithms for different clustered VLIW configuration and technology nodes. We specifically discern the benefits of a compiler based scheme as compared to a hardware only scheme and compare our results with some of the earlier algorithms. Readers interested in results in restricted but more realistic context of commercially available real clustered machines such as C6X are referred to some of our earlier work(28)(29).

It is important to mention here that the work and experimental results presented in this paper also focus on interconnect energy saving and integrated interconnect and functional unit energy savings. These results go significantly beyond some of the initial results presented in one of our earlier work(30) that focuses solely on scheduling to save energy in functional units. Additionally, in this paper, we also present results of savings offered by different algorithms across different technology nodes.

The rest of the paper is structured as follows. Section 2 describes the motivation for this work and presents some experimental evidences. Section 3 describes different scheduling algorithms for leakage energy management in functional units, energy optimization in interconnects, and the combined scheme to optimize energy in functional units as well as interconnects. Section 3 also describes the scheduler implementation in detail. Section 4 describes the scheduling algorithm with the help of examples. Section 5 describes our experimental setup, results, and a detailed analysis of results. Section 6 describes the related work in the area of scheduling for clustered architectures, energy aware scheduling for VLIW architectures, architectural approaches for leakage energy management, and efficient interconnect design. We conclude in section 7 with pointers to future directions.

2. Motivation

VLIW and clustered VLIW architectures are optimized for peak performance in order to meet real-time performance requirements of embedded applications. However, the functional units are underutilized due to the inherent variations in the ILP of the programs. The idleness is even more pronounced for a clustered VLIW architecture because of the contention for a limited number of slow interconnects which manifests itself in the form of many short idle cycles. The graph

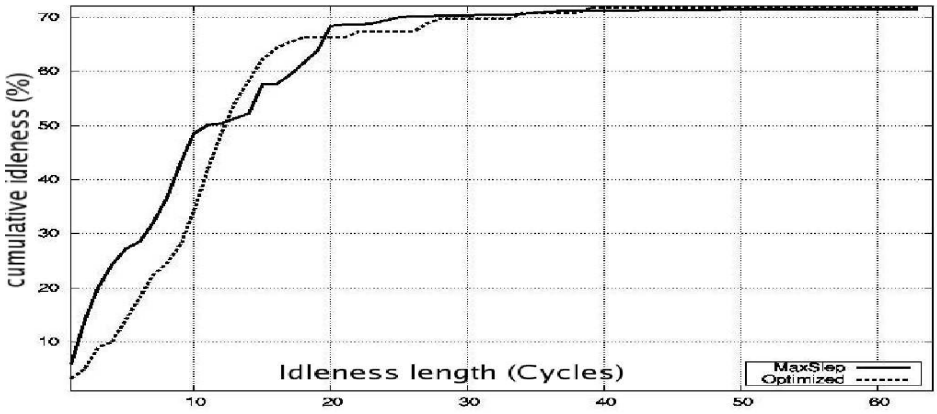


Figure 3: Functional Unit Idleness

titled 'Base' in Figure 3 shows the average cumulative distribution of idle cycles in integer ALUs for a 2-clustered machine having only 2 integer ALUs in each cluster and one bidirectional single cycle latency cross-path between clusters (details of our experimental setup and energy model appear in section 5). On an average, functional units are idle for 71% cycles in our collection of media benchmarks. Many small idle cycles constitute a large percentage of overall cycles. As Figure 3 depicts 50% of total 71% idle cycles have a duration less than or equal to 10 cycles in 'Base' architecture. We propose a scheme (Algorithm 1) that exploits instruction slack to aggregate the idleness in functional units by reducing the frequent transitions. The graph after applying our scheduling scheme is shown with title 'Optimized' in Figure 3. This shows that the many small idle cycles have been converted to large idle cycles by reducing transitions and only 34% of overall idle cycles are now less than 10 cycle. Idle cycle of length between 10 to 20 cycles constitute 32% of total idleness for 'Optimized' scheme while for the 'MaxSleep' scheme this is only 18%. This clearly shows that our scheme is able to exploit the slack to reduce the number of transitions thereby increasing the duration of idle periods.

Another way to reduce idle cycles (attributed to contentions for cross-paths) and thereby improve the performance is to use high-speed high-bandwidth cross-path for communication of data values among cluster. Previous studies have reported that the performance degrades by 12% when the latency of communication is doubled for a four clustered architecture, and that increasing the interconnection bandwidth from one to two improves the performance by as much as 10%(31). We

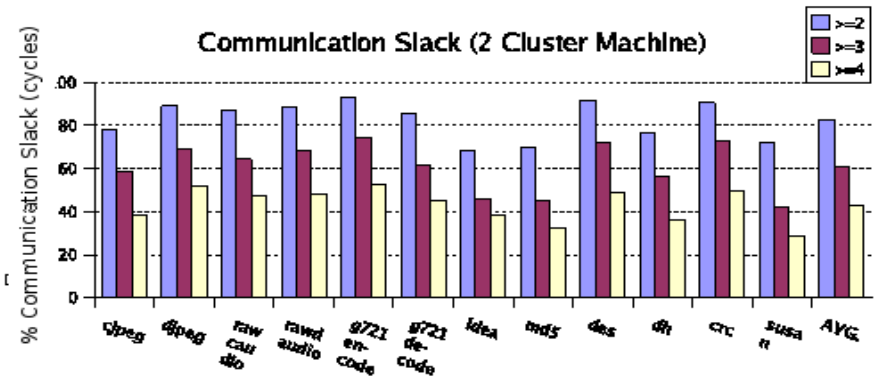


Figure 4: Communication Slack

also observe similar benefit in performance with approximately 15% reduction in idle cycles on an average by using two bidirectional single-cycle cross-paths between clusters as compared to the configuration which uses only one cross-path. However, having both the cross-path optimized for low latency results in high energy consumption in interconnects. This is because improving the latency of communication channel requires closely spaced repeaters which increase the area and energy overheads of repeaters(23). A high speed path for communication of data values among clusters indeed enables better performance, but we argue that not all data values are critical enough to be communicated on a high speed path, and that many communications are non-critical and can still happen on a slow path without affecting performance. *We define the communication slack of a data value on clustered architectures as the number of cycles between the time when the data value to be communicated becomes available (due to completion of execution of the producing instruction) and when the instruction that requires the data value is actually scheduled.* The available communication slack of a data value on clustered architecture is affected by data dependencies among instructions, limitation on the available number of functional units, and the limitations on the number of available cross-paths, their bandwidth, and the latency of cross-path communication. Figure 4 presents quantitative results to substantiate our arguments. This figure presents the percentage of required communication that has a slack of three cycles (two cycles and four cycles) or more for a two-cluster machine with two high speed bidirectional cross-paths between clusters. We observe that all the benchmarks have many communications with high slack values.

In particular *djpeg, g721encode, des, and crc* have 70% to 75% of communications with slack value of three cycles or higher. On an average, we observe that 60.88% (82.51% and 43.16%) of communications can sustain a latency of three cycles (two cycles and four cycles respectively) or higher.

Thus a more suitable option to reduce the idleness in functional units without incurring high energy overhead is to use interconnects between clusters with some paths optimized for latency and others for energy. We propose a scheduling mechanism (Algorithm 2) that exploits the communication slack to steer the non-critical communication over the slower but energy-efficient wires while assigning critical communication over the fast but more energy-consuming wires. Such a configuration which uses one bidirectional single-cycle cross-paths and one bi-direction three-cycle cross-path between clusters reduces the idle cycles by 13% on an average as compared to the configuration which uses only one cross-path.

Though a high bandwidth cross-path mitigates the contentions for cross-path and improve performance to some extent, the variation in ILP of the program coupled with cross-path contention still manifest itself in the form of many short idle cycle. The short idle cycles renders traditional leakage energy management schemes unusable. A hardware based scheme proposed in (27) utilizes the dual-threshold logic (and its capability of fast transition to and from low-leakage mode at moderate energy penalty) to perform leakage energy management for short idle cycles. However, the effective energy savings of this scheme is low because of high energy cost of fast transitions happening frequently. Our Transition aware scheduling scheme as described earlier exploits instruction slack to aggregate the idleness in functional units to improve the effective leakage energy savings by reducing the frequent transitions.

Thus, we propose a combined scheme (Algorithm 3) that exploits communication slack of data value and instruction slack together to reduce the energy in functional units and interconnects. The proposed scheme keeps the idle functional unit idle while maximizing the utilization of active functional units. At the same time, the proposed scheme exploits the communication slack of data value to utilize the low-power cross-path as much as possible.

3. The Scheduling Algorithm

The Elcor backend of the Trimaran infrastructure has a cycle scheduling algorithm designed and implemented for flat VLIW architectures(32)(33). We have modified this algorithm to perform leakage energy optimization for clustered VLIW architectures. Another loop has been added inside the main scheduling loop of

the cycle scheduler to perform cluster scheduling in an integrated fashion. The integrated approach(34)(35)(31) to cluster scheduling makes the cluster assignment decision during temporal scheduling. This is in contrast to phase-decoupled approaches(36)(37)(38) which perform cluster assignment prior to temporal scheduling. We propose three different scheduling algorithm. Algorithm 1 performs only leakage energy management assuming a machine model with low-leakage mode for functional units(30) but homogeneous interconnect (i.e. the energy cost of using any cross-path is same). Algorithm 2 performs interconnect energy optimization assuming a machine model with heterogeneous interconnects. However, this algorithm is more aggressive in interconnect energy management compared to one we proposed earlier(39) and is evaluated using a detailed interconnect energy model(40). Algorithm 3 is a new combined algorithm that simultaneously performs both leakage energy management in functional units as well as energy optimization in interconnect assuming the machine model having both low leakage mode for functional units as well as heterogeneous interconnects. However, it is important to note that combined Algorithm 3 gives preference to leakage energy management when exploiting slack and uses any left over slack for energy optimization in interconnects. This avoids excessive performance degradation as well as extra transitions in the functional units and associated energy overheads. Another reason we choose to give preference to functional units when exploiting slack is because its contribution to overall processor energy consumption is presumably more and it is one of the hot-spot and hence demands more aggressive energy saving. Table 1 presents summary of all algorithms. These algorithms differ with respect to cluster selection, functional unit binding, and cross-path assignment policies. However, the basic steps in all the algorithm are as follows.

Sr.	Name	Description	Machine Model
1	FuAlgo	Energy Efficient Scheduling for Functional Units(FUs)	FUs with low power modes
2	ICAlgo	Energy Efficient Scheduling for Interconnects(ICs)	ICs with varaition in latency and energy
3	FuICAlgo	Energy Efficient Scheduling for FUs and ICs (priority to FUs)	FUs with low power modes and ICs with varaition in latency and energy

Table 1: Summary of All Algorithms

1. Prioritizing the ready instructions
2. Assignment of a cluster to the selected instruction
3. Assignment of a functional unit to the selected instruction in the target cluster

4. Assignment of cross-paths for communicating the data values to the target cluster

In what follows, we describe how each of these step is performed and how different algorithms differ in their functioning.

3.1. Prioritizing the Ready Instructions

Instructions in the ReadyList are prioritized using a priority function that uses instruction slack and number of consumers of the instruction. Instructions with less slack should be scheduled early and are given higher priority over instruction with more slack to avoid unnecessary stretching of the schedule. Instructions with the same slack values are further ordered in the decreasing order of the number of consumers. An instruction with a large number of successors is more constrained in the sense that its spatial and temporal placement affects scheduling of more number of instructions and hence should be given higher priority. Giving preference to an instruction with many dependent instructions also enables better future scheduling decisions by uncovering a larger portion of the graph.

Scheduling slack of an instruction is defined as the difference between the earliest start time and the latest finish time of the instruction. Traditionally, slack is determined statically during dependence graph analysis before the scheduling begins, assuming a machine with infinite resources of each type. This calculation is inherently pessimistic as any real machine will have contentions for resources which prolongs the execution time. Since our algorithm exploits slack of instructions to delay their execution in order to save energy without affecting performance, a better quantification of available slack is of utmost importance. We quantify the slack of instructions while scheduling a region for the specific target machine by taking resource constraints into account. We first schedule the instruction using a simple cycle-by-cycle scheduler. The schedule time of the instructions is stored during this phase. In the second phase, this schedule time (termed as Late cycle) is used to determine the slack of the instruction. In our implementation, slack is dynamically updated for all the operations in the ready list after every cycle. The earliest schedule time of an instruction is set to the current cycle, before scheduling for the current cycle begin (Early cycle). The slack is then determined as a difference of the Early cycle and the Late cycle. Dynamic update of the slack after each cycle ensures that any consumed slack is taken into account while scheduling instructions in future cycles.

3.2. Cluster Assignment

Once an instruction has been selected for scheduling, we make a cluster assignment decision. The primary constraints are :

- The chosen cluster should have at least one free resource of the type needed to perform this operation
- Given the bandwidth of the channels among clusters and their usage, it should be possible to satisfy the communication needs of the operands of this instruction on the cluster by scheduling these communications in the earlier cycles (so that operands are available at the right time).

Selection of a cluster from the set of the feasible clusters is done based on two criteria. (1) to give preference to clusters that have an active functional unit to schedule the operation under consideration. (2) to give preference to cluster that reduces the overall cost of communication. Algorithm 1 (aimed at reducing energy consumption in functional units) uses criterion (1) as the primary criterion for cluster selection and criterion (2) is used only for tie-breaking. Algorithm 2 (aimed at reducing energy consumption in interconnects) uses criterion (2) for cluster selection. Algorithm 3 again uses criterion 1 as the primary criterion for cluster selection and criterion (2) is used as a secondary criterion. The communication cost is computed by determining the number and type of communications needed by a binding in the earlier cycles as well as the communication that will happen in the future. Future communications are determined by considering the successors of this instruction which have one of their parents bound on a cluster different from the cluster under consideration. This is due to the fact that if the instruction is bound to the cluster under consideration, it will surely lead to communication(s) in the future while scheduling the successors of the instructions. Although, we have experimented with many other heuristics for cluster assignment, the above mentioned heuristics seems to generate the best schedule in almost all cases from both the performance as well as energy perspective(41).

3.3. Functional Unit Binding

A functional unit binding scheme decides the binding of a chosen instruction to a functional unit and is important only in the context of algorithm 1 and 3. The algorithm maintains a FU map that explicitly keeps track of the status of each functional unit. A functional unit is marked to be in sleep mode after one cycle of idleness and activated on next use.

Algorithm 1 Energy Efficient Scheduling for Fus

```
Initialize ReadyList with root operations of the dependence graph of the region to be scheduled
CurrentCycle ← 1
while (ReadyList is not empty) do
    Initialize EarlyCycle with CurrentCycle, and LateCycle with SchedulingCycle determined using performance
    driven scheduling
    slack = LateCycle − EarlyCycle
    while (Not all operations in ReadyList have been tried once) do
        CurrentOperation ← UnSchedList.pop()
        ClusterPriority ← 1 //Schedule to reduce energy in FUs
        Target ← DetermineBestAlternative(CurrentOperation, CurrentCycle, ClusterPriority)
        if ((TargetCluster == −1) or (Slack ≥ SLACK_THRESHOLD and Target.Wakeup = 1)) then
            ReadyList.add(CurrentOperation)
            CONTINUE
        end if
        ScheduleFu(CurrentOp, Target.fu, Target.Cluster, CurrentCycle)
    end while
    CurrentCycle ← CurrentCycle + 1
    ReadyList.update()
end while
```

Algorithm 2 Energy Efficient Scheduling for ICs

```
Initialize ReadyList with root operations of the dependence graph of the region to be scheduled
CurrentCycle ← 0
while (ReadyList is not empty) do
    Initialize EarlyCycle with CurrentCycle, and LateCycle with SchedulingCycle determined using performance
    driven scheduling
    slack = LateCycle − EarlyCycle
    while (Not all operations in ReadyList have been tried once) do
        CurrentOperation ← UnSchedList.pop()
        ClusterPriority ← 0 //Schedule to reduce energy in ICs
        Target ← DetermineBestAlternative(CurrentOperation, CurrentCycle, ClusterPriority)
        if ((TargetCluster == −1) or (Slack ≥ SLACK_THRESHOLD and Target.CommEnergyCost ≥
        COMM_THRESHOLD)) then
            ReadyList.add(CurrentOperation)
            CONTINUE
        end if
        ScheduleComm(Target.CommOption)
        ScheduleFu(CurrentOp, Target.fu, Target.Cluster, CurrentCycle)
    end while
    CurrentCycle ← CurrentCycle + 1
    ReadyList.update()
end while
```

Algorithm 3 Energy Efficient Scheduling for Fus and ICs (priority to Fus)

```
Initialize ReadyList with root operations of the dependence graph of the region to be scheduled
CurrentCycle ← 1
while (ReadyList is not empty) do
    Initialize EarlyCycle with CurrentCycle, and LateCycle with SchedulingCycle determined using performance
    driven scheduling
    slack = LateCycle – EarlyCycle
    while (Not all operations in ReadyList have been tried once) do
        CurrentOperations ← UnSchedList.pop()
        ClusterPriority ← 1
        Target ← DetermineBestAlternative(CurrentOperation, CurrentCycle, ClusterPriority)
        if ((TargetCluster == −1) or (Slack ≥ SLACK_THRESHOLD and (Target.Wakeup == 1)) then
            ReadyList.add(CurrentOperation) //Schedule to reduce energy in FUs primarily
            CONTINUE
        end if
        ScheduleComm(Target.CommOption)
        ScheduleFu(CurrentOp, Target.fu, Target.Cluster, CurrentCycle)
    end while
    CurrentCycle ← CurrentCycle + 1
    ReadyList.update()
end while
```

If the functional unit required for the instruction under consideration is active in the target cluster, it is bound as usual. Otherwise, the available slack of the instruction is considered. If the slack is below a threshold (we use the threshold value of 0 in our experiment) the functional unit required by the instruction is woken up. In case there is more than one alternative available (for activating), the functional unit which is in sleep mode for a longer duration is woken up in order to amortize the cost of waking up. In case the instruction possesses enough slack, its scheduling is deferred to a future cycle and it is put back in the ReadyList. Note that the next time this instruction is picked up for scheduling, its earliest scheduling time and hence the slack gets updated. This guarantees that the slack of an instruction reduces monotonically and eventually comes below the threshold ensuring that it is scheduled. Hence the algorithm is guaranteed to terminate.

3.4. Cross-path binding

The cross-path assignment scheme is required only in the context of algorithm 2 and algorithm 3. The scheme is designed to minimize the energy consumption due to inter-cluster communication without affecting runtime performance. In order to meet this objective, the low power cross-paths are used in preference to the high power cross-paths wherever possible. More precisely, the assignment of cross-paths to communications is done as follow. To schedule a communication, its earliest scheduling cycle, latest scheduling cycle, and slack values are determined first. The earliest scheduling cycle for a communication is the cycle in

Procedure 4 DetermineBestAlternative

```
INPUT: ThisOp, ThisCycle, ClusterPriority
OUTPUT: Determines best scheduling alternative for this binding
FirstTarget.Cluster ← −1;
FirstTarget.CommCost ← 1000000;
FirstTarget.CommEnergyCost ← 1000000;
SecondTarget.Cluster ← −1
SecondTarget.CommCost ← 1000000;
SecondTarget.CommEnergyCost ← 1000000;
for (CurrentCluster ranging from FirstCluster through LastCluster) do
    if (FU required by ThisOp is available in ThisCycle for CurrentCluster) then
        if (Cross-paths required by ThisOp are available in ThisCycle for CurrentCluster) then
            CommAlternative ← DetermineBestCommAlternatives(ThisOperation, CurrentCluster, ThisCycle)
            FuAlternative ← DetermineBestFuAlternatives(ThisOperation, CurrentCluster, ThisCycle)
            if ((FU under consideration is in active mode) and (FirstTarget.CommEnergycost ≥ CommAlterna-
            tive.CommEnergyCost)) then
                FirstTarget.CommCost ← CommAlternative.CommCost
                FirstTarget.CommEnergyCost ← CommAlternative.CommEnergyCost
                FirstTarget.CommOption ← CommAlternative.CommOption
                FirstTarget.Cluster ← CurrentCluster
                FirstTarget.Fu ← FuAlternative.Fu
            else
                if (SecondTarget.CommEnergycost ≥ CommAlternative.CommEnergyCost) then
                    SecondTarget.CommCost ← CommAlternative.CommCost
                    SecondTarget.CommEnergyCost ← CommAlternative.CommEnergyCost
                    SecondTarget.CommOption ← CommAlternative.CommOption
                    SecondTarget.Cluster ← CurrentCluster
                    SecondTarget.Fu ← FuAlternative.FallBackFu
                end if
            end if
        end if
    end if
end for
if ((FirstTarget.cluster! = −1) and (ClusterPriority == 0 or FirstTarget.CommCost <=
SecondTarget.CommCos)) then
    FinalTarget ← FirstTarget
    FinalTarget.Wakeup ← 0
else
    FinalTarget ← SecondTarget
    FinalTarget.Wakeup ← 1
end if
RETURN FinalTarget
```

Procedure 5 DetermineBestFuAlternative

INPUT: ThisOp, ThisCluster, ThisCycle
OUTPUT: Best functional unit and fallback functional unit for this Binding
for (CurrentFu ranging from FirstFu through LastFu in ThisCluster) **do**
  if (CurrentFu is available in ThisCycle and Current Fu can execute ThisOp) **then**
    if (CurrentFu is active) **then**
      $Target.Fu \leftarrow CurrentFu$
      $Target.FallBackFu \leftarrow -1$
      RETURN Target
    else
      if ($CurrentFu.SleepPeriod > Target.SleepPeriod$) **then**
        $Target.FallBackFu \leftarrow CurrentFu$
        $Target.SleepPeriod \leftarrow CurrentFu.SleepPeriod$
      end if
    end if
  end if
end for
RETURN *Target*

Procedure 6 DetermineBestCommAlternative

INPUT: ThisOp, ThisCluster, ThisCycle
OUTPUT: Best communication alternatives and cost for this binding
for (All Communication required for scheduling ThisOp on ThisCluster in ThisCycle) **do**
  CurrentCommOption ← *DetermineBestCommOption*(*CurrentComm*)
  Target.CommOption.add(CurrentCommOption)
  Target.CommCost+=DetermineCommunicationCost(CurrentCommOption)
  Target.CommEnergyCost+=DetermineCommEnergyCost(CurrentCommOption)
end for
RETURN *Target*

Procedure 7 DetermineBestCommOption

INPUT : ThisComm
OUTPUT: A tuple CommOption (Comm, Cross) that associates best cross-path with ThisComm
CommOption.Comm ← ThisComm
Determine the EarlyCommCycle, LateCommCycle and the CommSlack for ThisComm
Determine the free minimum energy consuming cross-path, ***target_cross*** that can transfer CurrentComm between EarlyCommCycle and LateCommCycle
CommOption.Cross ← target_cross
RETURN *CommOption*

Procedure 8 ScheduleComm

INPUT: ThisCommOption
while (CurrentComm=ThisCommOption.pop()) **do**
  Schedule CurrentComm.Comm on CurrentComm.CrossPath on CurrentComm.ScheduleCycle
  Mark CurrentComm.CrossPath busy from CurrentComm.ScheduleCycle for CurrentComm.CrossPath.Latency Cycles
end while

Procedure 9 ScheduleFu

INPUT: ThisOp, ThisFu, ThisCluster, ThisCycle
Schedule ThisOp on ThisFu on ThisCluster on ThisCycle
Mark ThisFu in ThisCluster Busy from ThisCycle for ThisFu.Latency Cycles

which the data value to be communicated is produced in the source cluster, plus one. The latest scheduling time for communication is the scheduling cycle of first consuming instruction, minus one. The difference between the earliest scheduling cycle and the latest scheduling cycle is the communication slack. In order to avoid delaying the consuming instruction and the consequent possible stretch of the schedule, a communication is assigned to a least energy consuming cross-path that can transfer the data value within the available slack for communication. Thus the cross-path assignment scheme maximizes the usage of low power cross-paths subject to the availability of slack in the communication, and thus, as far as possible, performance degradation is minimized and energy saving is maximized.

Algorithm 2 uses a more aggressive scheme that exploits the instruction slack also and converts it to communication slack. This scheme defer an instruction that requires communications with total energy cost above communication threshold (COMM_THRESHOLD) if it possess a slack that is above a slack threshold (SLACK_THRESHOLD). The average energy cost of communications associated with a binding is determined according to following cost metric.

$$\begin{aligned} CommEnergyCost &= (A * \sum FastComm + B * \sum SlowComm \\ &+ C * \sum FutureComm)/TotalComm \end{aligned} \quad (1)$$

where *FastComm* represents number of the transfers that can happen on the fast cross path. *SlowComm* represents number of communications that can happen over the slow cross path and *FutureComm* represents the future communication that will happen on fast or slow cross-path depending on availability. *TotalComm* is the total number of communication that happen as a side effect of this binding. The selection of A, B, and C is architecture specific and depends on available communication options in a clustered architecture and their relative cost. We found that A=1.0, B=0.33 and C=0.67 work well in practice for the kind of heterogeneous interconnect we consider in our experiments. Of course, the weight can be chosen to reflect the heterogeneous interconnect under consideration. For our experiments, we have found that SLACK_THRESHOLD = 1 and COMM_THRESHOLD=.67 works well in practice. The above heuristic leads to scheduling the instruction under consideration at a later cycle preferably on a slow but more energy efficient cross-path. Again it is important to note that the next time this instruction is picked up for scheduling, its earliest scheduling time and hence the slack get updated. This guarantees that the slack of an instruction reduces monotonically and eventually comes below the threshold ensuring that it

is scheduled. Hence the algorithm is guaranteed to terminate. Finally, the combined Algorithm 3 exploits instruction slack only for leakage energy management in functional units (as Algorithm 1) and uses the communication slack only for interconnect energy optimization. Combined Algorithm 3 does not delay the instructions in order to seek opportunities to schedule communication on low power cross-path. Thus, it does not convert the instruction slack into communication slack to save energy in the interconnect.

3.5. Scheduler Implementation

The cycle scheduler as implemented in Trimaran(32) targets a flat VLIW class of architectures(33). It maintains a *ReadyList* of operations whose predecessors have already been scheduled. In each iteration of the main scheduling loop, the highest priority operation is selected from the *ReadyList* and scheduled in the current cycle if it satisfies all the resource constraints. In our implementation, slack and hence priority is recalculated at the beginning of each cycle for all the operations in the *ReadyList* using the schedule time determined in an earlier performance oriented scheduling pass. After taking the highest priority instruction, *DetermineBestAlternative* determines all scheduling alternatives of *CurrentOperation* in all clusters. *DetermineBestAlternative* scans each cluster one by one and check if the instruction under consideration can be scheduled in current cycle on cluster under consideration using some functional unit (though it can be in sleep mode) and available cross-path for requisite communication for this scheduling. It prioritizes clusters which match this feasibility criteria into two categories. *FirstTarget* stores the cluster that can accommodate the instruction under consideration in current cycle with minimum communication cost and also has an active functional unit. *SecondTarget* stores the cluster that can accommodate the instruction under consideration in current cycle with minimum communication cost but does not has an active functional unit. To make these decision, it uses procedure *DetermineBestFuAlternative* and *DetermineBestCommAlternative* to return the best functional unit for scheduling the instruction under consideration and best option for communicating the requisite data values for the scheduling under consideration and its associated cost. The best functional unit for performing the operation is one which is available and active. If no functional units is active, this procedure return the functional unit which is in sleep mode for longest time as a FallBackAlterNative. For determining the best alternative for communication, Procedure *DetermineBestCommAlternative* considers each required communication one by one, for each such communication it chooses the best communication option using procedure *DetermineBestCommOption*. Procedure *DetermineBestCommOp-*

tion returns the minimum energy consuming cross-path that is available and can schedule the communication under consideration within the available communication slack associated with communication under consideration. The procedure *DetermineBestCommOption* finally aggregate the cost of all communication so determined to calculate the overall cost of the binding. It also computes the average energy cost of the binding by averaging the energy cost of all requisite communication for this binding using cost function described earlier. Finally, the procedure *DetermineBestAlternative* decides between the Target clusters determined using above mentioned two criteria. Algorithm 2 uses *FirstTarget* or *SecondTarget* as *FinalTarget* depending on which one has the minimum communication cost. Algorithm 1 and Algorithm 3 uses *FirstTarget* if it is not empty and it uses *SecondTarget*, otherwise.

The top level scheduling algorithm proceeds after getting information about *FinalTarget*. Algorithm 1 puts back the instruction if it required waking up a resource in *TargetCluster* (i.e., there is no cluster available in the current cycle with an active functional unit to schedule the instruction) and instruction slack is above a threshold. Thus, this scheme exploits scheduling slack of instructions to save energy in function resources by keeping them idle for longer period of time while reducing transitions. The Algorithm 2 puts the instruction back in the *ReadyList* if the *FinalTarget.CommCost* is above a threshold and instruction also has enough slack. By doing this, it converts the instruction slack into communication slack in order to save energy in interconnects. However, if this is not the case the instruction is scheduled in *Target.Cluster* on *Target.Fu* using *Target.CommOption*. The procedures *ScheduleComm* and *ScheduleFu* associate the communication and computation resources for this binding and mark them busy. Finally, algorithm 3 performs cluster selection based on communication cost subject to availability of active resource and delays scheduling the instruction in the current cycle if it possesses enough slack and requires waking up a resource. Of course the cross-path assignment scheme still assigns the communications required by this instruction to slower cross-paths subject to availability and save energy in interconnects. However, the important difference between Algorithm 3 and Algorithm 2 is that unlike Algorithm 2 the instruction slack is not exploited to delay instructions to save energy in interconnect in Algorithm 3.

4. Example

In this subsection, we present two examples that illustrate how the available slack of instructions and communications is exploited by the proposed schedul-

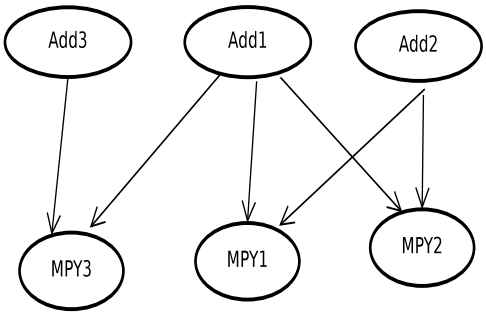


Figure 5: Dependence Graph 1

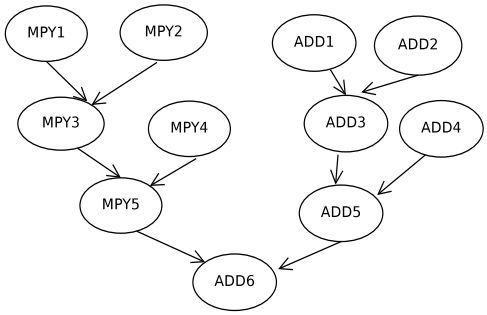


Figure 6: Dependence Graph 2

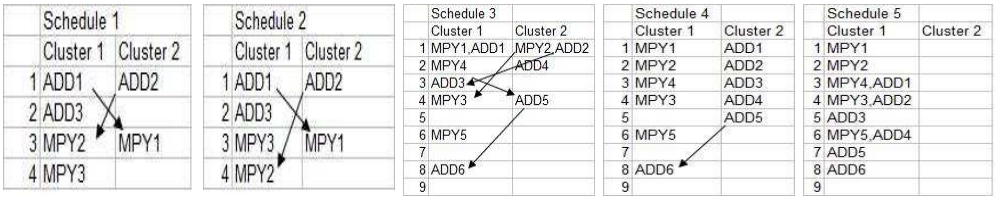


Figure 7: (a) Schedule 1 and 2 (for Graph 1) (b) Schedule 3,4 and 5 (for Graph 2)

ing algorithms 2 and 1 respectively to get energy benefits without hurting performance.

Figure 5 shows a portion of a data dependency graph and Figure 7 (a) shows two possible schedules for this dependency graph. We assume a two-clustered machine with each cluster having an adder, a multiplier and a fast communication bus. Schedule 1 has ADD1 and ADD2 scheduled on adders of cluster 1 and cluster 2 respectively in cycle 1. To perform multiplication, the results of these operations are transferred to the other cluster in cycle 2. The remaining addition operation ADD3 is also initiated in cycle 2 on cluster 1. The results of ADD1 and ADD2 can be used in cycle 3 on cluster 1 and cluster 2 respectively to perform MPY2 and MPY1 on multipliers. Though MPY3 does not require any inter-cluster communication, it is still executed in cluster 1 at cycle 4 because of non-availability of a multiplier in cycle 3. The scheduler decides to schedule MPY2 ahead of MPY3 in schedule 1 assuming that MPY2 is on the critical path. However, MPY3 gets preference if it is on the critical path as shown in schedule 2. Note that in this case, MPY2 needs to be scheduled in cycle 4 on cluster 1 again because cluster 1 has only one multiplier. *The important point to note here is that the scheduler when scheduling MPY2 in cycle 4 in cluster 2 has the knowledge that it can take two cycles to transfer the result of ADD2 over the communication channel without*

stretching the schedule. In such a situation if a slow but more energy-efficient bus is available, our scheduling algorithm 2 decide to steer communication to such a cross-path (as shown with darker arrow in schedule 2). Notably, even though three additions are ready to be scheduled in the first cycle only two of them can be scheduled (only two adders are available in this case). Similarly though the addition operations finish in opposite clusters in cycle one the results can not be utilized for multiplications in cycle 2 because it takes at least one cycle to transfer the results to the other clusters. This shows how contention among computation and communication resources in clustered architectures manifests itself in the form of greater computation and communication slack. Notably, the contention for resources is more in clustered architectures as compared to flat architectures because of distribution of resources. Our scheduling algorithm 2 leverages this increased slack and takes into consideration the criticality of an instruction and the available cycles to communicate requisite data values while scheduling an instruction in a given cycle. Accordingly, communication is assigned to the most energy-efficient cross-path that can transfer the value in the available communication cycles.

Figure 6 shows another dependence graph and Figure 7 (b) shows possible schedules for this dependency graph generated for a 2-clustered VLIW architecture having 1 adder and 1 multiplier in each cluster and a bidirectional bus between the two clusters with 1 cycle transfer latency. Schedule 3 is generated by a performance-oriented scheduler. Performance oriented scheduler schedules MPY1, and ADD1 in cluster 1 and MPY2, and ADD2 in cluster 2 in cycle 1 as shown in Schedule 3. The remaining ready operation MPY4 and ADD4 are scheduled in cycle 2 in Cluster 1 and Cluster 2 respectively. ADD3 (MPY3 resp.) is scheduled in cycle 3 (cycle 4 resp.) because it takes a cycle to transfer the result of ADD2 (MPY2 resp.) from cluster 2. Similarly MPY3 is scheduled in cycle 4 because of an extra cycle needed to transfer results from cluster 2. Consequently, the scheduling of MPY5 happens in cycle 6 whereas ADD5 is scheduled in cycle 5 in cluster 2. Finally ADD6 operation is scheduled in cycle 8 using the result of ADD5 from cluster 2. The Total schedule length is 9 cycles and total number of inter-cluster communication needed by the performance oriented scheduler based on eager (as early as possible) approach is 4.

Scheduling the same set of operations using our energy-efficient Schelling algorithm 1 generates schedule 4. The major point to note is that the scheduler leverages the available slack and map all the MPY operations (that are on the critical path) in cluster 1 while scheduling all add operations in parallel on cluster 2. The number of transitions from active mode to low leakage mode and vice-versa for M1, M2, A1, and A2 are 2, 2, 4, 4 for schedule 3 and 2, 0, 2, 2 for

schedule 4 respectively. Finally, schedule 4 is much more balanced. The resource usage vector of schedule 3 is (4,2,1,1,1,1,0,1,0) and that of the schedule 4 is (2,2,2,2,1,1,0,1,0). Cycle to cycle variation in resource usage is clearly reduced in schedule 4 as compared to schedule 3 that in turn helps in reducing step power (cycle to cycle variation in power) and peak power dissipation (maximum power consumed in a cycle)(42)[2] in addition to reducing transition energy overheads that maximize the leakage energy savings. Additionally, Schedule 4 reduces the number of inter-cluster communication to 1 whereas schedule 3 requires 4 inter-cluster communications.

Slight tweaking of our energy aware scheduling heuristic helps to make more conservative spreading of computation to map all the operation to just cluster 1 in case of this dependence graph, keeping cluster 2 completely idle as shown in schedule 5. This further saves even more leakage energy in some scheduling regions depending upon the dependencies among operations. The number of transitions from active mode to low leakage mode and vice-versa for M1, M2, A1, and A2 are 2, 0, 2, 0 for schedule 5. Schedule 5 is much more balanced. The resource usage vector of schedule 5 is (1,1,2,2,1,2,1,1,0). Finally, schedule 5 does not require any inter-cluster communication that helps to save even more energy.

5. Experimental Evaluation

5.1. Setup

We have used the Trimaran suite(32) for our experimentation. Trimaran was developed to conduct state-of-the-art research in compilation techniques for ILP architectures with a specific focus on VLIW class of architectures. We have modified the Trimaran suite to generate and simulate code for a variety of clustered VLIW configurations. The machine description module has been upgraded to describe various clustering related parameters such as the number of clusters, number and types of functional units in each cluster, interconnection network parameters such as number and types of buses between different clusters, and their latency parameters. These parameters are fed to the parameterized machine-dependent optimization modules in the backend. Major modifications have been performed in the Trimaran scheduler and register allocator module (which was originally written for a class of flat VLIW architectures) to faithfully account for

[2]It is noteworthy that the energy is total power consumed over all execution cycles and conversely average power is total energy consumed divided by total number of cycles(43)

the conflicts due to limitations on the number of available functional units and registers in a cluster as well as the limitations on the number of available cross-paths between clusters. The scheduler has been modified to implement the scheduling algorithm described in the last section. We have used twelve benchmarks out of which nine are from mediabench(44)(45) *(viz. cjpeg, djpeg, rawcaudio, rawdaudio, g721encode, g721decode, md5, des, and idea)*, two from netbench(46)(47) *(viz. crc, and dh)*, and one *(susan)* is from MiBench(48)(49). We have tried other benchmarks from these suits as well but these are the only ones which compiled successfully and executed correctly in the Trimaran framework and hence we report results for them.

We present results for an unclustered, a two-cluster machine and a four-cluster VLIW machine. The unclustered VLIW configuration has 4 ALUs, 2 load-store units, 1 branch unit, and 64 registers. The 2-clustered configuration has 2 ALUs, 1-load store units, 1 branch unit and 32 registers in each cluster, whereas the 4-clustered configuration has 1 ALU, 1-load store unit, 1 branch unit and 16 registers in each cluster. We consider two interconnect configuration named LL configuration and PP configuration. LL configuration has two fast cross-path between clustered allows transfer of two data values per cycles. LP configuration allows transfer of 2 data values 1 per cycle on L cross-path and 1 per 3 cycles on P cross-path. The number of functional units selected for the VLIW configurations are such that the performance achieved using this configuration is within 95% of the peak performance achieved by using many more functional units. This moderate number of functional resources guarantees that the benefits reported have not been obtained by trivially putting the numerous idle functional units into the low leakage mode. We report results only for Integer ALUs which are heavily used and pose a challenge for any leakage energy management scheme. Thus the benefits reported here have not been magnified by the leakage energy benefits of the load-store, branch, and FP units which are mostly idle. It is important to note that reducing the integer functional unit beyond this point for example reducing the number of integer functional units to one per cluster leads to drastic performance degradation (on an average close to 20%). This increase in execution time due to serialization of potentially parallel operations further aggravate the amount of inter-cluster communication required that in turn shows up as more idleness in the functional units which is favorably exploited by our algorithms (that is designed to thrive on even short idle cycles) for energy benefit.

5.2. Energy Model

We have used the same analytical energy model as in (27) to directly compare the functional unit energy benefits of the proposed schemes over the pure hardware based scheme proposed in(27). We briefly describe this model here. The reader is referred to (27) for details. The total energy in a functional unit in this model is determined as follows:

$$\mathbf{E'_{total}} = \mathbf{DynamicEnergy} + \mathbf{LeakageEnergy} + \mathbf{TransitionEnergy} + \mathbf{SleepModeEnergy}$$

$$\mathbf{E'_{total}} = \mathbf{n_A}(\alpha\mathbf{E_A} + (\mathbf{1}-\mathbf{D})\mathbf{E_{S_1}}) + (\mathbf{n_A D} + \mathbf{n_{UI}}) * (\alpha\mathbf{E_{s_0}} + (\mathbf{1}-\alpha)\mathbf{E_{s_1}}) + \mathbf{M_z}((\mathbf{1}-\alpha)\mathbf{E_A} + \mathbf{E_{Sleep}}) + \mathbf{n_Z E_{s0}}$$

Here $\mathbf{n_A}$ is the number of active cycles, $\mathbf{n_{UI}}$ is the number of uncontrolled idle cycles, $\mathbf{n_Z}$ is the number of sleep cycles and $\mathbf{M_z}$ is the number of transitions. We have determined these values differently for each configuration by using the trimaran simulator. $\mathbf{E_{s_0}}$ and $\mathbf{E_{s_1}}$ are low leakage and high leakage energy and are related by the following equations.
$\mathbf{E_{s_0}} = \mathbf{s} * \mathbf{E_{S_1}}, \mathbf{0.0001} \leq \mathbf{s} \leq \mathbf{0.01}$ and $\mathbf{E_{s_1}} = \mathbf{p} * \mathbf{E_A}, \mathbf{0} \leq \mathbf{p}$

Where $\mathbf{p}$ is the ratio of the maximum leakage energy expended to the maximum energy for evaluation per unit of time (1 cycle). After simplifying and normalizing the equations with respect to active energy, The following model for total energy consumption is obtained :

$$\mathbf{E_{total}} = \mathbf{n_A}(\alpha + (\mathbf{1}-\mathbf{D})\mathbf{p}) + (\mathbf{n_A D} + \mathbf{n_{UI}})(\alpha\mathbf{sp} + (\mathbf{1}-\alpha)\mathbf{p}) + \mathbf{M_z}((\mathbf{1}-\alpha) + \mathbf{E_{Sleep}}/\mathbf{E_A}) + \mathbf{n_Z sp}$$

The technology parameters that we have used (s=0.01 and $E_{Sleep}/E_A = 0.01$) are also the same as in (27) in order to compare the benefits of our scheduling algorithm to the hardware-only scheme. Considering the current 70nm fabrication technology where leakage energy is on par with dynamic energy, we set p to 0.5 .α is activity factor and $\mathbf{D}$ is the duty cycle of the clock. We use a typical value of 0.5 for both of these parameters in our simulation as in (27). For determining the energy benefits in interconnects we have used the INTACTE energy model(40).

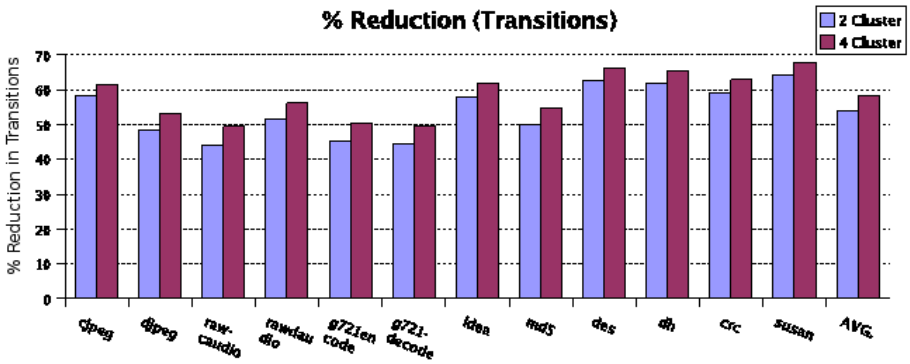


Figure 8: % Reduction in Transitions with scheduling w.r.t. H/W only Scheme for Algorithm 1

5.3. Results

We have performed a detailed experimental evaluation of the proposed algorithms in terms of execution time, energy benefits and overall energy delay product as well as impact of technology scaling on the energy benefits. The functional unit energy benefits are presented with respect to hardware only scheme that puts a functional unit into low leakage mode after one cycle of idleness. Figure 8 presents percentage reduction in number of transitions by using the scheduling algorithm 1 targeted to exploit instruction slack to reduce the unnecessary transitions and the associated energy benefit. Algorithm 1 reduces the number of transitions by 53.97%, and 58.29% as compared to hardware only scheme. Figure 9 presents the associated energy benefits of these reduced transitions for 2-Clustered and 4-Clustered machine using analytical energy model as described in last subsection. Algorithm 1 improves the functional unit energy consumption by 15.11% and 16.92% for 2-clustered and 4-clustered architecture respectively. The reduction in the number of transitions and achievable energy benefit depends on the total available slack in scheduling instructions as well as the distribution of idle cycles in the benchmark. Benchmarks like des, dh, crc, and susan have many short idle cycles and our algorithm is able to exploit the available slack in these applications to avoid many transitions. In the case of g721encode and g721decode, the available slack is relatively less and consequently the reduction is also less.

As compared to the performance oriented scheduler, the proposed algorithm suffers only a marginal performance loss of 0.3% and 0.5% in the context of 2-clustered and 4-clustered architecture. The reason for this performance loss is

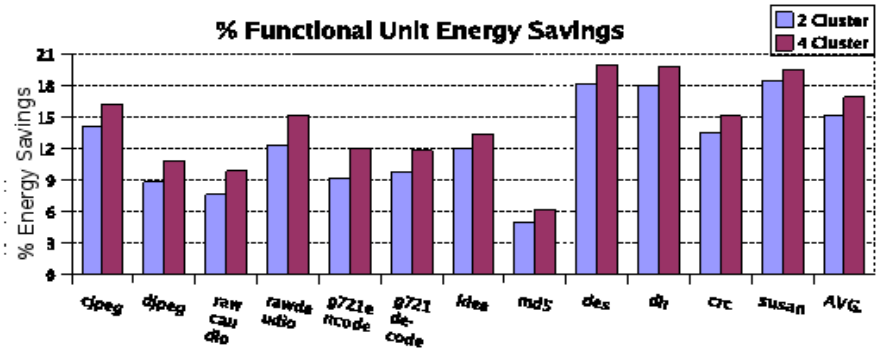


Figure 9: % Functional unit Energy Benefit w.r.t H/W only Scheme for Algorithm 1

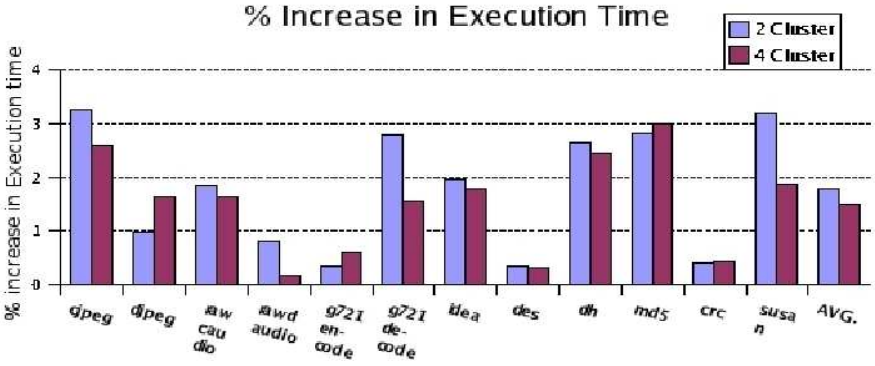


Figure 10: % Increase in Execution Time of LP conf. w.r.t. LL conf. for Algorithm 2

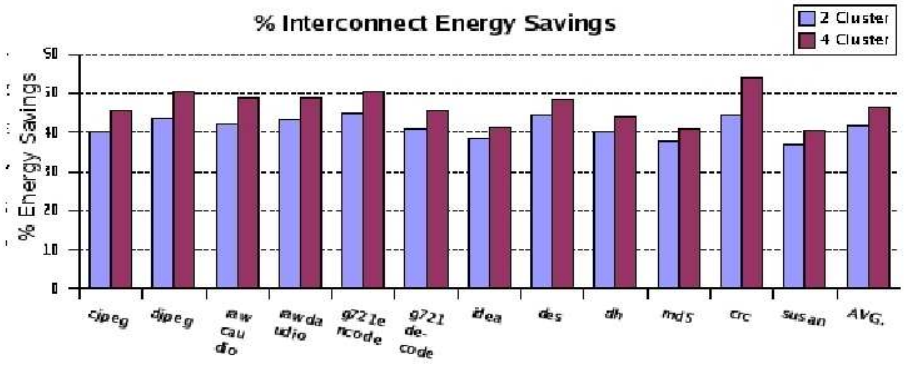


Figure 11: % Energy Benefit of LP conf. w.r.t. LL conf. for Algorithm 2

inherent inaccuracies in determining the available slack. Due to this, slack is sometime over-estimated which in certain cases lead to performance penalty due to serialization of operations. However, as the results show that it is rare and its overall effect on performance is only marginal.

The impact of technology on the benefits of our compiler directed leakage management scheme is depicted in Figure 12 which plots the benefit of our scheme on the top of hardware only scheme for three different technology nodes namely 90 nm, 65 nm, and 45 nm. The benefits of our scheme are even higher for technologies such as 90nm where is leakage is 20% to 30% of overall energy (we assume 25% with p=0.25 in this experiment). The benefit in even smaller technologies such as 45nm (assuming leakage is 65% with p=.65) are slightly less but still significant. The reason for these trend is as follows: Our scheme is actually geared towards reducing the transitions and associated energy overheads. Thus, when the overall contribution of leakage is more and corresponding dynamic energy contribution is less, the extra transitions and the impact of savings by reducing these transitions is also relatively less.

Figure 10 presents the percentage increase in execution time of using LP configuration (having 1 fast and 1 slow cross-path) with the proposed scheduling algorithm over LL configuration (having 2 fast cross-path). The average percentage increase in execution time is 1.8% and 1.5% for 2-clustered and 4-clustered machine respectively. The percentage energy benefit of using LP configuration scheduled using algorithm 2 as compared to LL configuration is 41.5% and 46.8% respectively for 2-clustered and 4-clustered machine as determined using the IN-

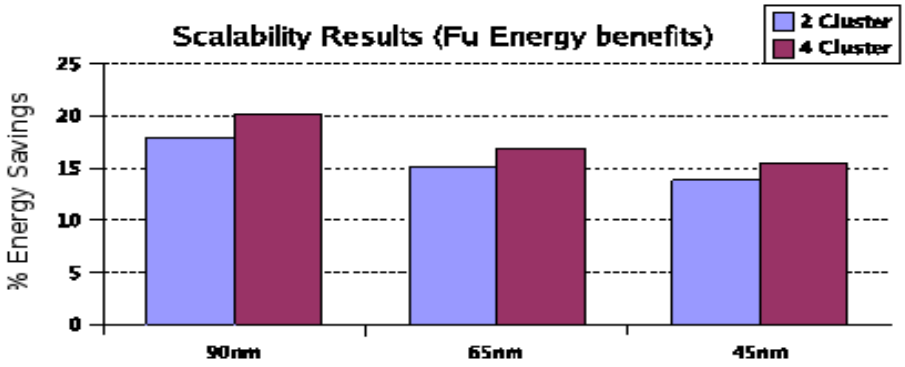


Figure 12: Scalability Results For Fu Energy Savings (90nm, 65nm, and 45nm) for Algorithm 1

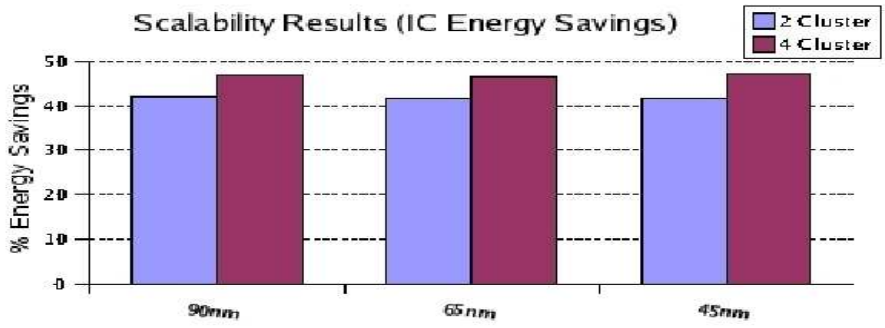


Figure 13: Scalability Results for IC Energy Savings (90nm ,65nm, and 45nm) for Algorithm 2

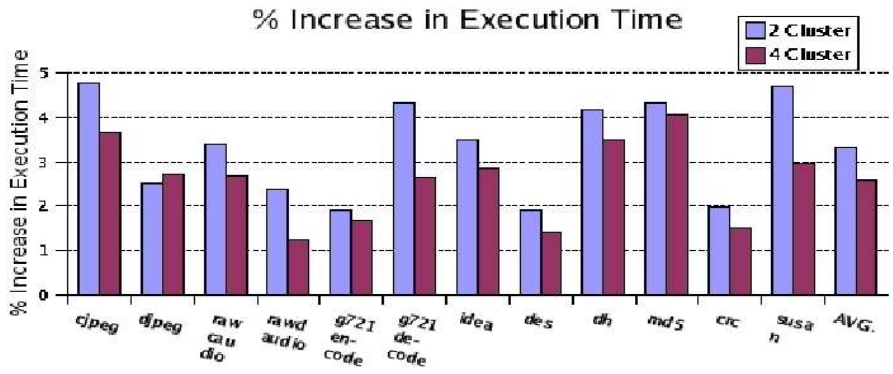


Figure 14: % Increase in Execution Time of LP conf. w.r.t. LL conf. for Algorithm 3

TACTE (40) interconnect energy model. Programs having more communications with high slack values *viz. djpeg, g721encode, des, and crc* suffer only a marginal performance degradation and gives significant energy benefit because many of the communication in these programs are scheduled on slow cross-path by the proposed scheduling algorithm 2. In contrast, programs with fewer communications with high slack values *viz. idea, md5, and susan* suffer a moderate performance degradation with the LP configuration and give relatively less energy benefits. However, a very small overall performance degradation occurs with the LP configuration whereas a significant amount of communication energy savings are obtained. This shows the effectiveness of our communication scheduling mechanism that selectively maps communications with high latency tolerance onto a high latency bus and communication with low latency tolerance to low latency bus.

Figure 13 depicts the impact of technology scaling on the benefit of our compiler scheduling scheme for heterogeneous interconnect for three different technology nodes namely 90nm 65nm and 45nm. The benefit of our scheme are roughly the same across different technology nodes with slight variations. This is because the smaller technologies though leads to increase in leakage part in some interconnect components such as repeaters, buffers, and flops (note that wire does not have the leakage component), there is also reduction in dynamic component of the energy in all the components including wires. This leads to roughly the same benefits of using the slower interconnect over faster interconnect across different technology nodes.

Figure 14 presents the percentage increase in execution time by applying al-

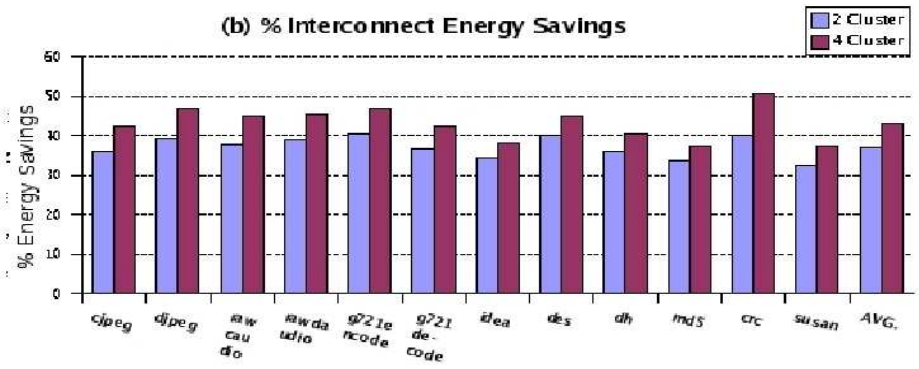


Figure 15: % Energy Benefit of LP conf. w.r.t. LL conf. for Algorithm 3

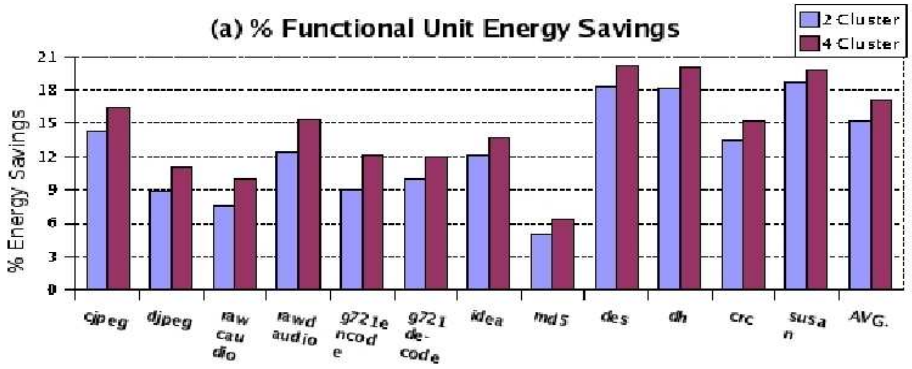


Figure 16: % Functional Unit Energy benefits of scheduling over H/W only scheme for Algorithm 3

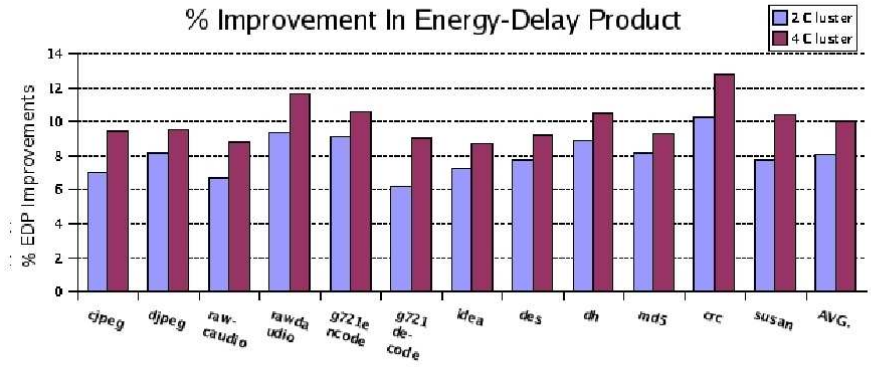


Figure 17: % Overall Benefit in EDP of scheduling Algorithm3 on LP conf. as compated to LL conf. and H/W only scheme for Fu Transitions

gorithm 3 that exploit the instruction slack to save leakage energy in functional units by reducing the transition and also migrates the communication with high slack value to slow bus. The percentage increase is presented w.r.t clustered architecture with LL configuration scheduled by performance oriented scheduler and have a hardware based scheme to optimize leakage energy in functional units(27). The average increase in execution time is 3.3% and 2.5% for 2-clustered and 4-clustered architectures which is higher than the average increase in execution time for algorithm 2 that only optimize the interconnect energy. This is because combined scheme uses the instruction slack for doing the leakage energy management in functional units and this leads to more cases where two communication simultaneously need the fast cross-path. In other words some of the instruction slack that was used up implicitly in Algorithm 2 is no longer available (because it is already used up for leakage energy management in functional units) and this shows up in the form of extra execution cycles in combined scheme.

Figure 15 shows the percentage saving in communication energy of scheduling Algorithm 3 on LP configuration as compared to LL configuration. Algorithm 3 saves the average communication energy by 37.1% and 43.1% which is slightly lesser than Algorithm 2 because the Algorithm 3 does not exploit instruction slack for saving communication energy explicitly and it gives priority to saving energy in functional units. However, there is still significant communication energy savings achieved by the combined algorithm attributed to available communication slack. Figure 16 presents the percentage savings in functional unit energy of combined algorithm as compared to hardware only scheme. The average saving in

functional unit energy is 15.3% and 17.2% for 2-Clustered and 4-Clustered machine respectively. As expected the percentage savings are roughly the same as Algorithm 1 because combined algorithm gives first priority to functional unit energy savings. The slight increase in functional unit energy is attributed to increase execution time with combined scheduling algorithm which is due to usage of slow cross-path in certain cases that leads to extra idleness in functional units.

Figure 17 gives the percentage savings in energy-delay product of processor by using combined scheme conservatively assuming that functional units constitute 30% of processor energy and interconnect constitute 20% of processor energy (because the actual figure can vary from system to system and has strong dependence on circuit, design style and technology parameters). We observe that even with this conservative assumption, the overall energy-delay product of the processor is improved on an average by 8% and 10% for 2-Clustered and 4-Clustered architecture which is a significant saving.

6. Related Work

In this section, we briefly describe the earlier work done in the area of instruction scheduling for clustered architectures, architectural approaches for leakage energy management, energy aware scheduling for VLIW architectures, and efficient cross-path design.

6.1. Instruction Scheduling for Clustered Architectures

Earlier proposals for scheduling on clustered VLIW architectures can be classified into two main categories, viz., phase-decoupled approaches and phase-coupled approaches. A phase-decoupled approach to scheduling works on a data flow graph (DFG) and performs partitioning of instructions into clusters to reduce inter-cluster communication while approximately balancing the load among clusters. The annotated DFG is then scheduled using a traditional list scheduler while adhering to earlier spatial decisions. A major argument in favor of this approach is that a partitioner having a global view of a DFG can perform a better job of reducing inter-cluster communication and load-balancing. The proposals in this direction are due to Desoli(37), Gonzalez(36), Lapinskii(38), Mahlke(50), Lee(51), and Nystrom(52). However, the phase-decoupled approach is known to suffer from the phase ordering problem. Since the spatial scheduler has only an approximate knowledge of load on clusters, usage of functional units, and cross-paths, approximate load-balancing often leads to cluster assignments which unnecessarily constrain the temporal scheduler in the later phase. Moreover, some

of these schemes are designed for reducing inter-cluster communication and end up reducing the ILP in the program in this pursuit(34)(31).

An integrated approach to scheduling combats the phase-ordering problem by combining spatial and temporal scheduling decisions in a single phase. The integrated approach considers instructions ready to be scheduled in a cycle and the available clusters in some priority order. The priority order for considering instructions is decided based on mobility, scheduling alternatives, the number of successors of an instruction etc. Similarly, the priority order for considering clusters is decided based on communication cost of assignment, earliest possible schedule time etc. An instruction is assigned a cluster to reduce communication or to schedule it at the earliest. The proposals in this direction are due to Ozer(34), Leupers(35), Kailas(31), and Nagpal(41)(53).

6.2. Architectural Approaches for Leakage Energy Management

Study of leakage energy management at the architectural level has mostly focused on storage structure such as cache. Yang et al., propose power supply gating of L1 cache cells(54). Kaxiras et al., dynamically adjust the interval after which a cache line is put into low leakage mode(55). Flaunter et al., propose a state-preserving drowsy cache design and a simple control scheme which is able to deliver most of the leakage energy benefits(56).

In contrast to storage structures, little work has been done on architecture level leakage energy management in the context of functional units. Our work directly improves over the work due to Albonesi et al. (27). This work proposes and evaluates an architectural policy for aggressively controlling leakage energy in integer ALUs. The 'MaxSleep' policy puts a functional unit into low leakage mode after one cycle of idleness. This scheme depends on dual threshold domino logic circuit with sleep mode proposed in (26) which has no delay penalty of transition between active mode and sleep mode. *Their performance evaluation using an analytical energy models in the context of spec benchmarks for superscalar architectures shows that for technology such as 65nm, the leakage energy benefit gained by such an aggressive scheme is significant. However the overhead of transitions from active mode into low-leakage mode and vice-versa are significant (on an average 30% when compared to a 'NoOverhead' scheme).*

6.3. Energy-Efficient Scheduling

Zhang et al.,(57) have proposed a rescheduling scheme to reduce dynamic and leakage energy in the functional units of a VLIW processor by exploiting the remnant slack of a performance-oriented schedule. In contrast Zhang et al., our ap-

proach works on raw unscheduled code with all the available slack for scheduling and directly exploits all the available slack thereby complementing any hardware based mechanism for leakage energy management. Kim et al.,(21) have proposed a leakage energy management scheme for VLIW processors that approximates the ILP available in the program using heuristics (as the exact estimation problem is itself NP complete). The calculation is done at the loop level granularity assuming that there is little variation in the ILP within the loop. Their scheme keeps only canonical subset of functional units that is sufficient to exploit this approximated ILP active. In contrast, our approach adaptively applies leakage energy management at a finer granularity based on available ILP. Gupta et al.,(58) propose a novel data structure called power-aware flow graph. Their leakage energy management scheme in the context of superscalar processors works over this graph to determine larger program regions called power blocks which offer opportunities to save leakage energy. ISA and architectural support is needed to switch on and off the functional unit at the boundaries of power blocks and nullify spurious on-off. Kim et al.,(42) have proposed a modulo scheduling algorithm that produces a more balanced schedule for software pipelined loops with an objective to reduce the peak power and step power dissipation. Though our algorithm is not directly designed towards improving the peak power and step power dissipation, it generates a more balanced schedule. This is because it tries to keep minimum number of functional unit active and try to use the active functional units as much as possible while keeping the idle functional units idle for longer durations(30).

6.4. Efficient Cross-path Design

As compared to reducing energy consumption in function blocks, study of energy efficiency in interconnects is still in its infancy. Previous work has concentrated on improving latency for interconnects in the context of distributed architectures. Gonzalez et al.(59) have evaluated different kinds of interconnects with different topologies and concluded that a point-to-point interconnect with an effective steering scheme is more efficient than a bus-based interconnect. Their experimental results also demonstrate that an asynchronous interconnect offers a performance comparable to an idealized interconnect at a low hardware implementation cost. Terechko et al.(12) has proposed various inter-cluster communication models for clustered architecture and perform a quantitative analysis to compare their benefits.

Closest to our proposal is the work by Balasubramonian et al.(25). They have also used the same interconnect energy model as proposed in (24) to evaluate techniques such as cache pipelining, exploiting narrow bit-width operands, and

interconnect load balancing in the context of superscalar architectures with heterogeneous interconnect. In contrast, our work is more focused on how communication slack in the context of clustered VLIW architecture can be exploited to gain the energy benefits and to explore the energy-performance trade-off while going from a BASE architecture to different configurations of clustered VLIW architectures. *Our results demonstrate that compile-time instruction scheduling utilizing a larger view of program can combine the instruction scheduling and communication scheduling in a profitable manner. On the other hand, a architecture with dynamic scheduling suffers from the problem of limited program view and incurs overheads and complexities of extra hardware for exploiting heterogeneous interconnects at run-time.* Thus, the choice of a heterogeneous interconnect is more suitable and beneficial for statically scheduled VLIW architectures as compared to dynamically scheduled architectures. Moreover, our results are based on a detailed and verified model of interconnect energy estimation whereas their study is mostly based on guesstimates based on earlier studies on interconnect energy estimation.

7. Conclusions and Future directions

In this work, we have proposed energy-aware instruction scheduling algorithms that exploits instruction slack and communication slack to save energy in two major energy hungry components of clustered VLIW architecture namely functional units and interconnects. We also proposed combined scheduling algorithm that simultaneously save energy in functional units and interconnect. A detailed experimental evaluation using trimaran framework confirms that proposed schemes are capable of providing significant energy savings thereby improving the usability of clustered architectures specifically in smaller technologies. Our compiler assisted leakage energy management scheme for functional units reduce the energy consumption of functional units approximately by 15% and 17% in the context of a 2-clustered and a 4-clustered VLIW architecture respectively with negligible performance degradation on the top of a hardware-only scheme. The interconnect energy optimization scheme improves the energy consumption of interconnect on an average by 41% and 46% for a 2-clustered and a 4-clustered machine respectively with 2% and 1.5% performance degradation. The combined scheme obtains slightly better energy benefit in functional units and 37% and 43% energy benefit in interconnect with slightly higher performance degradation. Even with the conservative estimates of contribution of functional unit and interconnect to overall processor energy consumption, the proposed combined scheme

obtains on an average 8% and 10% improvement in overall energy-delay product with 3.5% and 2% performance degradation for a 2-clustered and a 4-clustered machine respectively. In future, we are interested in evaluating the temperature benefit of the proposed scheme which is becoming more and more important in smaller technologies.

[1] D. Matzke, Will Physical Scalability Sabotage Performance Gains, IEEE Computer.

[2] T. N. Mudge, Power: A First Class Design Constraint for Future Architecture and Automation, in: HiPC '00: Proceedings of the 7th International Conference on High Performance Computing, Springer-Verlag, London, UK, 2000, pp. 215–224.

[3] D. Sylvester, H. Kaul, Power-Driven Challenges in Nanometer Design, IEEE Design and Test of Computers 18 (6) (2001) 12–22. doi:http://doi.ieeecomputersociety.org/10.1109/54.970420.

[4] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, Clustered Instruction-level Parallel Processors, Tech. rep., Hewlett-Packard (1998).

[5] J. M. P. R. Canal, A. Gonzalez, Dynamic cluster assignment mechanisms, in: Proc. of Sixth IEEE Intl. Symp. on High Performance Computer Architecture, 2000.

[6] G. S. Sohi, S. E. Breach, T. N. Vijaykumar, Multiscalar processors, in: 25 Years ISCA: Retrospectives and Reprints, 1998, pp. 521–532.

[7] U. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, B. Khailany, The imagine stream processor, in: Proc. of 2002 IEEE Intl. Conf. on Computer Design: VLSI in Computers and Processors (ICCD'02), IEEE Computer Society, Washington, DC, USA, 2002, p. 282.

[8] P. Marcuello, A. Gonzalez, Clustered speculative multithreaded processors, in: ICS '99: Proc. of 13th Intl. Conf. on Supercomputing, ACM Press, New York, NY, USA, 1999, pp. 365–372. doi:http://doi.acm.org/10.1145/305138.305214.

[9] J. E. Smith, Instruction-level distributed processing, Computer 34 (4) (2001) 59–65. doi:http://dx.doi.org/10.1109/2.917541.

[10] A. Capitanio, N. Dutt, A. Nicolau, Partitioned register files for VLIWs: a preliminary analysis of tradeoffs, in: Proceedings of the 25th annual international symposium on Microarchitecture, IEEE Computer Society Press, 1992, pp. 292–300. doi:http://doi.acm.org/10.1145/144953.145839.

[11] K. I. Farkas, P. Chow, N. P. Jouppi, Z. Vranesic, The multicluster architecture: reducing cycle time through partitioning, in: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, IEEE Computer Society, 1997, pp. 149–159.

[12] A. Terechko, E. L. Thenaff, M. Garg, J. V. Eijndhoven, H. Corporaal, Inter-Cluster Communication Models for Clustered VLIW Processors, in: Proc. of Intl. Symp. on High-Performance Computer Architecture, 2003, p. 354.

[13] Texas Instruments Inc., TMS320C6000 CPU and Instruction Set reference Guide, http://www.ti.com/sc/docs/products/dsp/c6000/index.htm (1998).

[14] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, F. Homewood, Lx: A Technology Platform for Customizable VLIW Embedded Processing, in: Proc. of 27th annual Intl. Symp. on Computer architecture, 2000, pp. 203–213.

[15] J. Fridman, Z. Greefield, The TigerSHARC DSP architecture, IEEE Micro (2000) 66–76.

[16] G. G. Pechanek, S. Vassiliadis, The ManArray Embedded Processor Architecture, in: Proc. of Euromicro Conf., 2000, pp. 348–355.

[17] J. Derby, J. Moreno, A High-performance Embedded DSP Core with Novel SIMD Features, in: Proc. of 2003 Intl. Conf. on Acoustics, Speech, and Signal Processing, 2003.

[18] OMAP5, http://www.ti.com/ww/en/omap/omap5/omap5-platform.html.

[19] N. Seshan, High VelociTI processing, IEEE Signal Processing Society 15 (1998) 86–101.

[20] R. Ho, K. Mai, M. Horowitz, The Future of Wires, Proc. of IEEE 89 (4) (2001) 490–504.

[21] H. S. Kim, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, Adapting Instruction Level Parallelism for Optimizing Leakage in VLIW Architectures, in:

Proc. of Conf. on Language, Compiler, and Tool for Embedded Systems, 2003, pp. 275–283.

[22] J. A. Butts, G. S. Sohi, A Static Power Model for Architects, in: MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture, ACM Press, New York, NY, USA, 2000, pp. 191–201. doi:http://doi.acm.org/10.1145/360128.360148.

[23] K. Banerjee, A. Mehrotra, A Power-Optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs, in: Proc. of IEEE Transactions on Electron Devices, 2002, pp. 2001–2007.

[24] M. L. Mui, K. Banerjee, A. Mehrotra, A Global Interconnect Optimization Scheme for Nanometer Scale VLSI with Implications for Latency, Bandwidth and Power Dissipation, in: IEEE Transactions on Electron Devices, 2004, pp. 195–203.

[25] R. Balasubramonian, N. Muralimanohar, K. Ramani, V. Venkatachalapathy, Microarchitectural Wire Management for Performance and Power in Partitioned Architectures, in: Proc. of Intl. Symp. on High-Performance Computer Architecture, 2005, pp. 28–39.

[26] V. Kursun, E. G. Friedman, Low swing Dual Threshold Voltage Domino Logic, in: GLSVLSI '02: Proceedings of the 12th ACM Great Lakes symposium on VLSI, ACM Press, New York, NY, USA, 2002, pp. 47–52. doi:http://doi.acm.org/10.1145/505306.505317.

[27] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, E. G. Friedman, Managing static leakage energy in microprocessor functional units, in: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture, IEEE Computer Society Press, Los Alamitos, CA, USA, 2002, pp. 321–332.

[28] R. Nagpal, Y. N. Srikant, Integrated temporal and spatial scheduling for extended operand clustered vliw processors, in: Conf. Computing Frontiers, 2004, pp. 457–470.

[29] R. Nagpal, Y. N. Srikant, A graph matching based integrated scheduling framework for clustered vliw processors, in: ICPP Workshops, 2004, pp. 530–537.

[30] R. Nagpal, Y. N. Srikant, Compiler-assisted leakage energy optimization for clustered vliw architectures, in: EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software, ACM, New York, NY, USA, 2006, pp. 233–241. doi:http://doi.acm.org/10.1145/1176887.1176921.

[31] K. Kailas, A. Agrawala, K. Ebcioglu, CARS: A New Code Generation Framework for Clustered ILP Processors, in: Proc. of 7th Intl. Symp. on High-Performance Computer Architecture, 2001, p. 133.

[32] Trimaran System, http://www.trimaran.org/.

[33] S. G. Abraham, W. M. Meleis, I. D. Baev, Efficient Backtracking Instruction Schedulers, in: Proc. of Intl. Conf. on Parallel Architectures and Compilation Techniques, 2000, pp. 301–308.

[34] E. Ozer, S. Banerjia, T. M. Conte, Unified Assign and Schedule: A New Approach to Scheduling for Clustered Register File Microarchitectures, in: Proc. of Intl. Symp. on Microarchitecture, 1998, pp. 308–315.

[35] R. Leupers, Instruction scheduling for clustered VLIW DSPs, in: PACT '00: Proc. of 2000 Intl. Conf. on Parallel Architectures and Compilation Techniques, IEEE Computer Society, Washington, DC, USA, 2000, p. 291.

[36] A. Aleta, J. M. Codina, J. Sanchez, A. Gonzalez, Graph-partitioning based Instruction Scheduling for Clustered Processors, in: Proc. of Intl. Symp. on Microarchitecture, 2001, pp. 150–159.

[37] G. Desoli, Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach, Technical Report, Hewlett-Packard (1998).

[38] V. S. Lapinskii, M. F. Jacome, G. A. De Veciana, Cluster Assignment for High-Performance Embedded VLIW processors, ACM Trans. on Design and Automation of Electronic Systems (2002) 430–454.

[39] R. Nagpal, Y. N. Srikant, Exploring energy-performance trade-offs for heterogeneous interconnect clustered vliw processors., in: Proc. of Intl. Conf. on High Performance Computing, 2006, pp. 497–508.

[40] R. Nagpal, A. Madan, A. Bharadwaj, Y. N. Srikant, INTACTE: An Interconnect Area, Delay, and Energy Estimation Tool for Microarchitectural Explorations, in: Proceedings of the international conference on compilers, architecture, and synthesis for embedded systems, ACM Press, New York, NY, USA, 2007.

[41] R. Nagpal, Y. N. Srikant, Integrated Temporal and Spatial Scheduling for Extended Operand Clustered VLIW Processors, in: Proc. of Conf. on computing frontiers, 2004, pp. 457–470.

[42] H. Yun, J. Kim, Power-aware Modulo Scheduling for High-Performance VLIW Processors, in: Proc. of 2001 Intl. Symp. on Low Power Electronics and Design, ACM Press, 2001, pp. 40–45.

[43] V. Venkatachalam, M. Franz, Power reduction techniques for microprocessor systems, ACM Computing Survey 37 (2005) 195–237. doi:http://doi.acm.org/10.1145/1108956.1108957.
URL http://doi.acm.org/10.1145/1108956.1108957

[44] C. Lee, M. Potkonjak, W. H. Mangione-Smith, MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, Intl. Symp. on Microarchitecture.

[45] MediaBench, http://cares.icsl.ucla.edu/MediaBench/.

[46] B. M.-S. Gokhan Memic, W. Hu, NetBench: A Benchmarking Suit for Network Processor, CARES Technical Report.

[47] NetBench, http://cares.icsl.ucla.edu/NetBench/.

[48] J. R. Matthew Guthaus, D. Ernst, MiBench: A Free, Commercially Representative Embedded Benchmark Suite, IEEE 4th Annual Workshop on Workload Characterization.

[49] MiBench, http://www.eecs.umich.edu/mibench/.

[50] M. Chu, K. Fan, S. Mahlke, Region-based Hierarchical Operation Partitioning for Multicluster Processors, SIGPLAN Notices (2003) 300–311.

[51] W. Lee, D. Puppin, S. Swenson, S. Amarasinghe, Convergent Scheduling, in: Proc. of Intl. Symp. on Microarchitecture, 2002, pp. 111–122.

[52] E. Nystrom, A. E. Eichenberger, Effective Cluster Assignment for Modulo Scheduling, in: Proc. of 31st annual ACM/IEEE Intl. Symp. on Microarchitecture, IEEE Computer Society Press, 1998, pp. 103–114.

[53] R. Nagpal, Y. N. Srikant, A Graph Matching Based Integrated Scheduling Framework for Clustered VLIW Processors, in: Proc. of ICPP Workshop on Compile and Runtime Techniques Parallel Computing, 2004, pp. 530–537.

[54] S.-H. Yang, B. Falsafi, M. D. Powell, K. Roy, T. N. Vijaykumar, An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches, in: HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture, IEEE Computer Society, Washington, DC, USA, 2001, p. 147.

[55] S. Kaxiras, Z. Hu, M. Martonosi, Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power, in: ISCA '01: Proceedings of the 28th annual international symposium on Computer architecture, ACM Press, New York, NY, USA, 2001, pp. 240–251. doi:http://doi.acm.org/10.1145/379240.379268.

[56] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, T. Mudge, Drowsy Caches: Simple Techniques for Reducing Leakage Power, in: ISCA '02: Proceedings of the 29th annual international symposium on Computer architecture, IEEE Computer Society, Washington, DC, USA, 2002, pp. 148–157.

[57] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte, Y.-F. Tsai, Exploiting VLIW Schedule Slacks for Dynamic and Leakage Energy Reduction, in: Proc. of Intl. Symp. on Microarchitecture, 2001, pp. 102–113.

[58] S. Rele, S. Pande, S. Onder, R. Gupta, Optimizing Static Power Dissipation by Functional Units in Superscalar Processors, in: Proc. of 11th Intl. Conf. on Compiler Construction, 2002, pp. 261–275.

[59] A. G. Joan-Manuel Parcerisa, Julio Sahuquillo, J. Duato, Efficient Interconnects for Clustered Microarchitectures, in: Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques, 2002, pp. 291–300.