A Graph Matching Based Integrated Scheduling Framework for Clustered VLIW Processors

Rahul Nagpal and Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science
Bangalore, India
{rahul,srikant} @csa.iisc.ernet.in

Abstract

Scheduling for clustered architectures involves spatial concerns (where to schedule) as well as temporal concerns (when to schedule) and various clustered VLIW configurations, connectivity types, and inter-cluster communication models present different performance trade-offs to a scheduler. The scheduler is responsible for resolving the conflicting requirements of exploiting the parallelism offered by the hardware and limiting the communication among clusters to achieve better performance without stretching the overall schedule.

This paper proposes a generic graph matching based framework that resolves the phase-ordering and fixed-ordering problems associated with scheduling on a clustered VLIW processor by simultaneously considering various scheduling alternatives of instructions. We observe approximately 16% and 28% improvement in the performance over an earlier integrated scheme and a phase-decoupled scheme respectively without extra code size penalty.

1 Introduction

Clustering has been proposed to overcome the scalability problem associated with centralized VLIW architectures and to make them suitable for use in embedded systems[7]. A clustered VLIW architecture has more than one register file and connects only a subset of functional units to a register file. Groups of small computation clusters can be fully or partially connected using either a *point-to-point* network or a *bus-based* network. Many inter-cluster communication (ICC) models such as send-receive model, extended operand model, extended result model and broadcast model are possible[20]. Clustering avoids area and power consumption problems of centralized register file architectures while retaining high clock speed. High clock speed can be leveraged to get better performance but this demands high quality partitioning of operations among clusters because communication among clusters is limited and slow due to technological constraints[13]. A compiler (scheduler) for these architectures is responsible for binding operations to resources in different clusters. Different clustered datapath configurations, connectivity types, and ICC models proposed in literature[20] present different architectural constraints and performance trade-offs to the scheduler. The scheduler is required to resolve the conflicting goals of exploiting hardware parallelism as well as minimizing communication among clusters.

The earlier proposals for scheduling clustered VLIW architecture fall into two main categories namely phase-decoupled approach and the integrated approach. The phase-decoupled approach of scheduling[4][6][10] partitions a DFG of instructions into clusters to reduce ICC while approximately balancing the load among clusters. The partitioned DFG is then scheduled using a traditional list scheduler while adhering to earlier spatial decisions. Proponents of this approach argue that a partitioner having a global view of the DFG can perform a good job of reducing ICC. However pre-partitioning schemes in general suffer from the phase-ordering problem. A spatial scheduler has only an approximate knowledge (if any) of the usage of cross-paths, functional units, and load on clusters. This inexact knowledge often leads to spatial decisions which may unnecessarily constrain a temporal scheduler and produces a suboptimal schedule. Though reducing ICC can be the right approach while scheduling on a clustered VLIW processor with only explicit move instructions for ICC, it may not produce a good schedule in the case of other ICC models.

An integrated approach[15][16][12][9] towards scheduling considers the instructions ready to be scheduled in a cycle and the available clusters in some priority order. Priority orders for considering instructions are often based on parameters such as mobility, scheduling alternatives and

Proceedings of the 2004 International Conference on Parallel Processing Workshops (ICPPW'04)
1530-2016/04 $20.00 © 2004 **IEEE**

number of successors of an instruction. Priority orders for considering clusters are based on parameters such as communication cost of assignment and earliest possible schedule time. An instruction is assigned a cluster either to reduce communication or to schedule it at the earliest. Usually scheduling alternatives for an instruction depend on resource and communication requirements of other instructions ready to be scheduled in the current cycle. Thus schemes which follow a fixed statically determined order for considering instructions and clusters suffer from what we call as *fixed-ordering problem* and often end up producing suboptimal schedule in terms of performance, code size or both. The proposed techniques in literature are also very specific about a particular architectural configuration and ICC model and demand major variations to achieve optimal performance for a different model. Since design and validation of a scheduler is a complicated and time consuming task, a generic scheduler which can accommodate architecture specific constraints and can easily adapted to different architectural variations is preferable.

This paper proposes a generic graph matching based framework that resolves the phase-ordering and fixed-ordering problems associated with scheduling a clustered VLIW datapath by simultaneously considering various temporal and spatial scheduling alternatives of instructions. The framework considers the possible mapping of instructions to resources in different clusters and can easily accommodate architecture specific constraints typically found in spatial architectures. it requires only slight tweaking of heuristics to get optimal performance for different clustered VLIW configurations and ICC models. We have implemented this framework for the state-of-art Texas instruments TMS320C64X DSP architecture[18]. We evaluate its effectiveness in improving performance. The rest of the paper is organized as follows. Section 2 describes the target machine model. Section 3 contains our graph matching based scheduling framework. Section 4 presents performance statistics and a detailed evaluation based on experimental data. In section 5, we briefly mention related work in the area. We conclude in section 6. A detailed example demonstrating the benefits of graph matching based scheduler can be found in associated technical report[14].

2 Problem Description

2.1 Machine Model

We consider a generic clustered machine model based on recently proposed clustered architectures. In our machine model a clusters can be homogeneous having identical functional units and register file (like in VelociTI architecture[18]) or heterogeneous with different cluster having a different mix of functional units and register file (like in HP Lx architecture[8]). The connectivity among

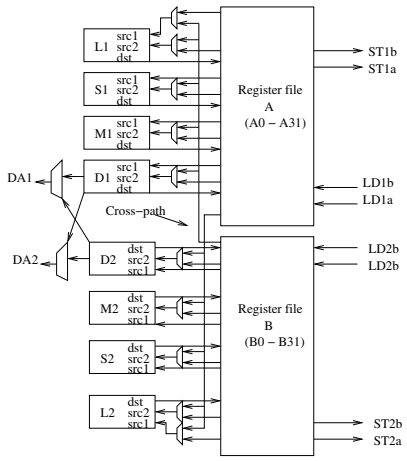

Figure 1. TMS320C64X CPU Data Path

clusters can be full or partial. The functional units can vary in terms of the their operational and communicative capabilities. An operation can be performed on more than one resource and a resource can perform more than one kind of operation. Some resources may also have some specialized task delegated to them. The inter-cluster communication model can vary. The architecture may even have a hybrid communication model where some of the functional units can communicate by snooping (reading) operands from the register file of a different cluster *without any extra delay* (as in VelociTI architecture[18]), while communication among some of the clusters is possible only through an explicit MV operation (as in HP Lx architecture[8]). Snooping capabilities of functional units can be varied in terms of operands a particular functional unit can snoop as well as particular clusters with which a function unit can communicate using the snooping facility. Our machine model also incorporates architecture-specific constraints typically found in clustered architectures. For example, some operations can be performed only on some specialized resources due to performance or correctness concerns. This machine model enables us to design a generic and pragmatic framework which can accommodate architecture-specific constraints and can be easily adapted to a variety of clustered architectures differing in datapath configurations and/or communication models. Next we briefly explain the TMS320C64X architecture, a production clustered processor having some of the features of our machine model, which we have used for practically evaluation of the framework.

TMS320C64X is a load-store RISC-style architecture (refer to figure 1). The architecture follows a homogeneous clustering philosophy and has two clusters (named A and B). Each cluster has a 32x32 register file and 4 functional units (named L, S, M, and D). In addition each data path has a cross-path to read operands from the other file and the address from one data path can be used to load and store values in the other data path. The ICC is restricted to two inter-

cluster move one in each direction per cycle. Functional units can snoop their operands from other cluster without any extra cost. However, there are some restrictions. L unit can read either of its operands from the other cluster while S, M, and D can read only their second source operand from the other cluster. Most common operations are possible on four to eight units and some units have specialized operations. Explicit move operation between two clusters can be performed by blocking one of L, S, or D unit for one cycle. All functional units have a single cycle latency and can initiate a new instruction every cycle. Most of the C64x instructions are of unit latency.

3 The Algorithm

```
input: A dataflow graph G
output: A triple (slot,cluster,unit) for each node in G
var
ready_list : List of nodes (instructions)
L=G.findScheduleLatencyForBaseVLIW();
G.preProcess(L);
ready_list.init(G)
while (!ready_list.is_empty()) do
  M=createMatchingGraph(readylist)
  C=M.solveMinimumCostILP()
  S=M.solveMaxNodeILP(C)
  while (!S.is_empty()) do
    E=S.remove()
    (i,u,c) = (E.node, E.unit,E.cluster)
    if (!rejectBasedOnCommunicationAndMobility(i))
    then
      if explicitMvNeeded(i,u,c) then
        scheduleExplicitMv(i,c)
      end if
      schedule instruction i on unit u in cluster c
    end if
  end while
  readylist.update(G)
  advanceCycle()
end while
```

Figure 2. Graph Matching based spatial and temporal scheduling algorithm

An outline of graph matching based scheduler is given in figure 2. The algorithm differs from the list scheduling because it considers all possible scheduling alternatives for all the instructions simultaneously and thus is able to remove the sub-optimality due to possible interference of scheduling alternatives of instructions. Graph matching based scheduler creates a bipartite graph that consists of a set of instruction nodes including all the instructions in the ready list and a set of resource nodes including all the resources in all the clusters. An edge connecting an instruction node and a resource node represents a possible scheduling alternative for the instruction. Each edge is assigned a cost determined by a cost function and the information regarding usage of cross-paths. This is followed by finding a minimal cost maximum node matching for this graph. Instruction are considered for scheduling using the alternative dictated by the selected match. To further reduce the communication and extra code, some of the instructions in the final match incurring high communication overheads but possessing enough scheduling freedom can be delayed for consideration in later cycles. Next we describe each of these steps in detail in separate subsections

3.1 Measuring Instruction freedom

Freedom available in scheduling an instruction is defined as the difference between the latest and the earliest time an instruction can be scheduled without stretching the overall schedule length. In general values of instructions freedom are calculated assuming an infinite resource machine thus ignoring all the resource constraints because the exact measurement of of instruction freedom in a resource constrained scenario is equivalent to finding an optimal schedule and hence is NP-complete. However, this leads to a very pessimistic calculation of the freedom. For a better estimation, we first schedule on a unclustered base VLIW processor with the same data path configuration but ignoring all communication delays. Since in general the best schedule that can be obtained on a clustered VLIW processor will be of the same length as that of the base VLIW processor, these freedom values incorporating resource constraints can be used without any disadvantage while scheduling on the clustered VLIW processors.

3.2 The Graph Construction

The graph construction process creates a bipartite matching graph that consists of a set of instruction nodes including all the instructions in the ready list and a set of resource nodes including all the resources in all the clusters. An instruction node is connected with a resource node if it is possible to schedule the instruction on the resource. Each edge is associated with a cost computed using the cost function and a communication vector which are described later. There can be more than one edge between an instruction and a resource depending on possible alternatives for cross-cluster communication. We add all the edges explicitly to the graph and later select the best alternative for the set of instructions ready to be scheduled. To make the graph complete, dummy edges are introduced between instruction nodes and resource nodes that do not have even a single edge between them. Dummy edges have infinite value and zero entries for the cost and communication vector respectively and are added to make it feasible to formulate the ILP.

3.3 ILP formulation of graph matching problem

Once all the possible scheduling alternatives for instructions ready to be scheduled in the current cycle are encoded

in the matching graph, we face the problem of finding a feasible minimum cost match while scheduling as many instructions as possible in the current cycle. Since we are not aware of any polynomial algorithm for solving the minimal cost maximal matching with the additional cross-path usage constraints, we formulate the problem as a sequence of two ILP problems. The first ILP problem finds a feasible minimum cost match. Since different factors determining the cost of binding are changed dynamically to make scheduling decisions with precise information available at the time of scheduling a set of instructions and the communication cost factor also becomes accurate from one cycle to another (as more and more parents of successors becomes scheduled), the cost of a real edge can be equal to sum of two or more edges. The second ILP problem maximizes the no. of non-dummy nodes matched for the minimum cost obtained by solving the earlier problem. The formulation and description of both ILP problems is as follows.

I	Set of ready instructions
i	An individual instruction
R	Set of all resources {L1,S1,D1,M1,L2,S2,D2,M2}
r	An individual resources
A(i,r)	Set of all alternative for scheduling instruction i on resource r
a	An individual alternative
M_{ir}^a	Boolean Match variable for scheduling i on r with alternative a
c_{ir}^a	Cost of scheduling i on r with alternative a
x_{irp}^a	Cross path usage for p^{th} cross-path while scheduling i on r with alternative a
N_p	Bandwidth of p^{th} cross-path
X	Set of all cross-paths
c_{max}	Cost of dummy edge
C	Cost of matching
N^R	Number of real edges in the matching

The objective function and feasibility constraints for the first problem can be stated as follows:

1. Minimize the cost of final match
2. Every instruction is assigned to at most one resource or no resource at all
3. All the resources are assigned some instruction or other (guaranteed because the graph is complete)
4. Cross-path usage of the final match does not exceed the cross-path bandwidth for any of the cross-path

formally

minimize

$$C = \sum_{i \in I,\, r \in R,\, a \in A(i,r)} M_{ir}^a * c_{ir}^a$$

subject to:

$$\forall i \in I \sum_{a \in A(i,r),\, r \in R} M_{ir}^a \leq 1$$

$$\forall r \in R \sum_{i \in I,\, a \in A(i,r)} M_{ir}^a = 1$$

$$\forall p \in X \sum_{i \in I,\, r \in R,\, a \in A(i,r)} M_{ir}^a * x_{irp}^a <= N_p$$

The objective function and feasibility constraints for the second problem can be stated as follows:

1. Maximize the no of real edges in the final match
2. Total cost of final match is not more than minimum cost of matching obtained by solving the first ILP
3. Every instruction is assigned to at most one resource or no resource at all
4. All the resources are assigned some instruction or other (guaranteed because the graph is complete)
5. Cross-path usage of the final match does not exceed the cross-path bandwidth for any of the cross-path

formally

maximize

$$N^R = \sum_{i \in I,\, r \in R,\, a \in A(i,r) \,\wedge c_{ir}^a < c_{max}} M_{ir}^a$$

subject to:

$$\sum_{i \in I,\, r \in R,\, a \in A(i,r)} M_{ir}^a * c_{ir}^a \leq C$$

$$\forall i \in I \sum_{a \in A(i,r),\, r \in R} M_{ir}^a \leq 1$$

$$\forall r \in R \sum_{i \in I,\, a \in A(i,r)} M_{ir}^a = 1$$

$$\forall p \in X \sum_{i \in I,\, r \in R,\, a \in A(i,r)} M_{ir}^a * x_{irp}^a <= N_p$$

3.4 The Cost Function

The cost function computes the cost of scheduling an instruction on a resource using a given communication alternative. Some important parameters deciding the cost are mobility of the instruction, number and type of communication required, and uncovering factor of the instruction. We define *uncovering factor* of an instruction as the number of instructions dependent on this instruction. As we mentioned earlier, a better estimate of the mobility of instructions is obtained by first scheduling on a base VLIW configuration since mobility of an instruction plays an important part in making scheduling decisions for the instruction. The mobility of an instruction is further reduced by $\alpha * (current\_time - VST)$ when an instruction is entered into the ready list, where VST is the time of scheduling this instruction on a base VLIW configuration. This reduction helps in further refining the estimate of freedom available in scheduling the instruction by taking into accounts the delay introduced due to ICC in the partial schedule generated so far. After each cycle the mobility of each instruction left in the ready list is reduced by a factor β to reflect the exact freedom left in scheduling each instruction in the next

cycle. The communication cost models proposed here is generic enough to handle different ICC models or even a hybrid communication model where communication among clusters is possible by combination of different ICC models. The communication cost is computed by determining the number and type of communication needed by a binding alternative. Communication vector is composed of triple *(explicit_mv,current_comm,future_comm)* which encodes the cross-cluster communication requirements of scheduling the instruction for the alternative under consideration. *explicit_mv* take care of communications that can be due to non-availability of snooping facility in a particular cycle or in the architecture itself. *current_comm* stands for usage of limited snooping capability if available in the hardware as in the case of extended operand processors. This facility can be used to reduce the code size and delays by reading some operands directly from register file of other clusters. *future_comm* is predicted by determining those successors of the current instruction which have some of their parents bound on a cluster different from one under consideration.

The final cost function combines together the cost due to various factors. Since the problem has been formulated as a minimum cost matching problem, alternatives with lesser cost are given priority over alternatives with higher cost. Communication factor is used to resolve resource conflict among alternatives having same mobility, while the mobility is used to decide among alternatives having same communication cost. Uncovering factor is given relatively little contribution and is used for tie-breaking in the cost function proposed. The mobility of an instruction reduces from one cycle to another to reflect the exact freedom available in scheduling an instruction. As the available freedom in scheduling an instruction reduces, its likelihood for selection in the final match increases. Once the mobility drops below zero, instead of adding to the cost of an alternative it starts reducing the cost. Thus for instruction which have passed all the freedom, the communication cost become less important and this further helps to increase their selection possibility. As more and more parents of successors of an instructions got scheduled, the communication cost factor become more accurate and its become more likely that instruction will be scheduled in a cluster that reduces the need for high future communication.

Figure 3 presents one set of values that we have used during our experiments. The given values are for an architecture that has limited snooping facility available and thus the *current_comm* is given half the cost of the *explicit_mv* required. *future_comm* is also assigned a lesser cost optimistically assuming that most of them will be accommodated in the free-of-cost communication slot. Clustered architectures mostly differ in the inter-cluster configuration and ICC models used and the cost function can be tweaked to reflect the associated trade-offs in the target architecture. Thus the cost function is generic and require slight tweaking for working with an architecture accommodating a different ICC model.

```
comm_cost=(X*current_comm*+Y*future_comm+
            Z*explicit_mv)/(MAX_COMM+1)
uncover_cost=(MAX_UNCOVER-uncovering_factor)/
             (MAX_UNCOVER-MIN_UNCOVER+1)
mob_cost=mobility/(MAX_MOB+1)
cost=A*mob_cost + B*comm_cost+C*uncover_cost
where:
MAX_UNCOVER : Maximum of uncovering factors of
                  instructions in the ready list
MIN_UNCOVER : Minimum of uncovering factors of
                  instructions in the ready list
MAX_MOB : Maximum of mobilities of
                instructions in the ready list
MAX_COMM : Maximum of communication
                costs of any of the alternatives
Constants :
α=1.00, β=1.00, X=0.5, Y=0.75, Z=1.00
A=1.00, B=1.00, C=0.10
```

Figure 3. Cost Function

3.5 Selective Rejection Mechanism

Since our algorithm schedules as many instructions as possible in each cycle, some of the instructions with high communication overheads may appear in the final match depending on the resource and communication requirements of other instructions being considered. For example if there is only one ready instruction for a particular resource, the algorithm will always decide to schedule it in the current cycle even if it possesses enough freedom and has high communication overheads. Thus in order to further reduce the inter-cluster communication and associated overheads, scheduling of instructions having high communication cost and enough freedom can be deferred to future cycles in the hope of scheduling them later using a less costly alternative. The uncovering factor can also be used to decide the candidate to be rejected in a better way. Thus whereas the ILP formulation is geared towards maximizing IPC, the selective rejection mechanism prune the final ILP solution for a cycle by selectively deferring instructions incurring high communication cost and possessing enough mobility to explore the trade-offs in code size and performance.

4 Experimental Evaluation

4.1 Setup

We have used the SUIF compiler[2] along with MACH-SUIF library[1] for our experimentation. We perform a number of classical optimizations to get a highly optimized intermediate representation. We generate code for TMS320C64X[21]. Ours is a hand-crafted code generator

Table 1. Details of the Benchmark programs

Name	Description	Category	L/N
AUTOCORR	auto correlation	Filters	.15
CORR 3x3	3x3 correlation with rounding	Filters	.11
FIR	Finite impulse response filter	Filters	.12
FIRCX	Complex Finite impulse response Filter	Filters	.09
IIRCAS	Cascaded biquad IIR FILTER	Filters	.18
GOURAUD	Gouraud Shading	Imaging	.46
MAD	Minimum Absolute Difference	Imaging	.17
QUANTIZE	Matrix Quantization w/ Rounding	Imaging	.38
WAVELET	1D Wavelet Transform	Imaging	.16
IDCT	IEEE 1180 Compliant IDCT	Transform	.09
FFT	Fast fourier transform	Transform	.19
BITREV	Bit Reversal	Transform	.31
VITERBI	Viterbi v32 pstn trellis decoder	Telecom	.17
CONVDOC	Convolution Decoder	Telecom	.20

Table 2. Legend

Legends	Meaning
GM	Graph Matching without Selective Rejection
GS	Graph Matching with Selective Rejection
UAS	Unified Assign and Schedule
MWP	Magnitude Weighted Predecessors Ordering
CWP	Completion Weighted Predecessors Ordering
UC	UAS with CWP
UM	UAS with MWP
LP	Lapinskii pre-partitioning Algorithm

which is designed to take advantage of the specialized addressing modes of the architecture[19]. Code generation is followed by a phase of peephole optimization and this highly optimized code is passed to our scheduler. We have interfaced our scheduler with the CPLEX ILP solver[3] to handle the ILP formulation of the graph matching problem. The scheduler annotates each instruction with the time-slot, cluster, and functional unit information. Register allocation is performed on the scheduled code using priority based graph coloring[5]. Any spill code that is generated is scheduled in separate cycles. After adding procedure prologue and epilogue, we emit the scheduled assembly code in a format acceptable to the TI assembler. After assembly and linking, we carry out simulation of the generated code using the TI simulator for the TMS320C64X architecture[21].

Table 1 summarizes the key characteristics of benchmark programs that we have used for experimental evaluation of our framework and a comparison with the pre-partitioning algorithm[10] and UAS[16]. These benchmarks are mostly unrolled inner loop kernels of MEDIABENCH[11], a representative benchmark of multimedia and communication applications and is specially designed for embedded applications. Table 1 also mentions the L/N ratio for each benchmark, where L is the critical path length and N is the number of nodes in the DFG for each kernel. L/N ratio is an approximate measure of the available ILP in the program. When L/N is high, the partitioning algorithm has little room to improve the schedule. *Programs with low L/N exhibit enough ILP for a careful partitioning and effective functional unit binding algorithm to do well.* The selected programs have L/N ratios varying between .09 to .46.

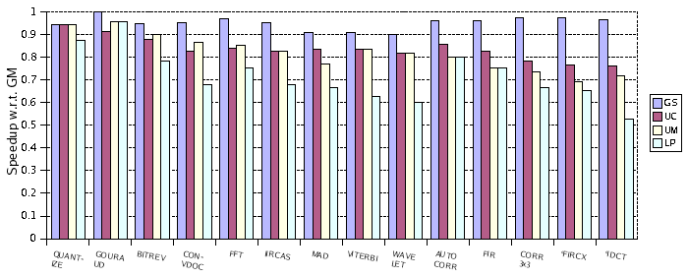

Figure 4. Speedup as compared to GM

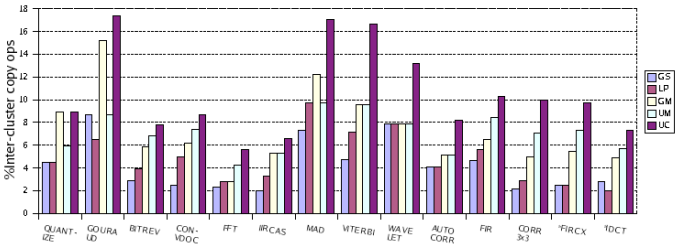

Figure 5. % Distribution of Explicit Inter-cluster MV Instructions vs. Other Instructions

4.2 Performance Statistics

Figures 4 depicts the speed-up of different algorithms as compared to graph matching based scheduling algorithm. We have experimented with two main heuristics for considering clusters namely completion weighted predecessors (CWP) and magnitude weighted predecessors (MWP) as proposed in the paper by Ozer et al.[16]. GM attains approximately 16.37%, 18.16%, and 28.51% improvement respectively over UC, UM, and LP. GS could attain approximately 11.94%, 13.82%, and 24.87% improvement respectively on the average on UC, UM, and LP. GS suffers a slight performance degradation of about 4.97% on the average compared to GM. Figure 5 presents the percentage distribution of explicit inter-cluster MV instructions vs. other instructions for different algorithms. On the average, the percentage of explicit inter-cluster MVs vs other instructions is 4.21%, 4.84%, 7.20%, 7.08%, and 10.52% for GS, LP, GM, UM, and UC respectively.

Graph matching based scheduler could attain significant speed-up over UC, UM, and LP on program exhibiting high ILP (low L/N). Moderate and marginal speed-up is observed over programs with medium and little amount of ILP. This further reinforces the fact that as the amount of parallelism available in the program increases, the optimal schedule length is dictated by the resource constraints and effective partitioning and functional unit binding become more important. The graph matching based scheduler performs best in terms of execution time. However, it introduces comparatively more explicit MV operation. GS reduces the no. of explicit MV operation with some performance degradation. GS and LP perform better than other algorithms in terms of extra code added. LP having a global view of DFG and tending to reduce ICC incurs less code size penalty. GM is

better than UM and UC and UC performs worse of all.

4.3 Performance Evaluation

As mentioned earlier we consider a generic machine model where resources vary in terms of their operational and communicative capabilities and ICC facility is a function of a resource rather than that of communicating clusters. On such a model, the resource and communication constraints are tightly coupled and are precisely known only while scheduling. Thus, the pre-partitioning algorithms trying to balance the load based on an approximate (or no) knowledge of the above facts and tending to reduce inter-cluster communication make spatial decisions which artificially constrain the temporal scheduler in the later phase. These algorithms thus suffer from the the well known phase-ordering problem. From experimental results, it is evident that this effect becomes prominent as the available ILP in the program increases. Though these algorithms may able to reduce the inter-cluster communication by working on a global view of the DFG, we observe that it is often done at the cost of performance.

UAS integrates cluster assignment into the list scheduling algorithm and shows improved performance over phase decoupled scheduling approaches, Ozer et al. have proposed many orders for considering clusters. However, the paper by Ozer et al.[16] does not propose any particular order for considering instructions in the ready queue. However, the order in which instructions are considered for scheduling has an impact on the final schedule. The integrated algorithms proposed earlier follow a fixed order for considering instructions and clusters and thus these algorithm suffer from the *fixed-ordering problem*. Earlier integrated algorithms in general and UAS in particular do not consider any future communication that may arise due to a binding and this may lead to a stretched schedule because of more communication than what the available bandwidth can accommodate in a cycle and hence more number of explicit move operations. These are the reasons why UAS suffers from performance and code size penalty. Another drawback of UAS is due to the fact that it does not consider functional unit binding. However, effective functional unit binding is important in the case of the machine model under consideration because resources vary in their operational and communicative capabilities and resource and communication constraints are tightly coupled.

GM improves over earlier integrated algorithms by considering all the possible scheduling alternatives obtained by varying communication options, spatial locations, and resource usage simultaneously instead of following a fixed order. GM resolves fixed-ordering problem by simultaneously selecting among scheduling alternatives of instructions with an aid of cost function (composed of various dynamically varying factors) while exploiting the communication facility and parallelism offered by the hardware. Since GM schedules as many instructions as possible in each cycle, some of the instructions with high communication overheads may appear in the final match depending on the resource and communication requirements of other instructions being considered. In order to further reduce the inter-cluster communication and extra code added, GS incorporates selective rejection mechanism to defer the scheduling of instructions having high communication cost and enough freedom to future cycles in the hope of scheduling them later using a less costly alternative.

5 Earlier Work

Integrated scheduling is described in Ozer[16], Leupers[12], and Kailas[9] while the phase decoupled schemes are due to Gonzalez[4], Desoli[6] and Lapinskii[10]. In what follows, we briefly describe proposals due to Lapinskii et al.[10] and Ozer et al.[16] which we have used for comparison with our framework.

Ozer et al.[16] have proposed an algorithm called unified-assign-and-schedule (UAS). UAS extends the list scheduling algorithm with a cluster assignment decision while scheduling. After picking the highest priority node, it considers clusters in some priority order and checks if the operation can be scheduled in a cluster along with any communication needed due to this binding. They have proposed various ways of ordering clusters for consideration such as no ordering, random ordering, magnitude weighted predecessors (MWP), and completion weighted predecessors (CWP). MWP ordering considers no of flow-dependent predecessors assigned to each cluster for operation under consideration. The instruction can be assigned to a cluster having the majority of predecessors. CWP ordering assigns each cluster a latest ready time for operation under consideration depending upon the cycle in which the predecessor operations produce the result. The clusters which produce the source operand of the operation late are given preference over others. An ideal hypothetical cluster machine model has been used and performance is based on statically measured schedule length.

To assure fairness in comparison we have modified UAS to give it the benefits of an extended operand ICC model. To get the the benefit of limited free-of-cost communication facility available in extended operand ICC model, we snoop the operand wherever possible to avoid unnecessary MV operation.

Lapinskii et al.[10] have proposed an effective binding algorithm for clustered VLIW processors. Their algorithm performs spatial scheduling of instructions among clusters and relies on a list scheduling algorithm to carry out temporal scheduling. Instructions are ordered for consideration using an ordering function consisting of as late as possible scheduling time of an instruction, mobility of an instruction

and number of successors of an instruction. They compute the cost of allocating an instruction to a cluster using a cost function

Their cost function considers the load on the resources and buses to assign the nodes to clusters. The load on functional units and communication channels is calculated by adapting the force directed scheduling approach[17] of behavioral synthesis. They emphasize that their partition algorithm strives to exploit the parallelism while minimizing inter-cluster communication is a secondary criterion. They have also proposed two binding improvement functions for further improving the schedule at some computational expense. These improvement functions reconsider nodes at the boundary of a partition and look for opportunities for improvement. Though they have proposed a good cost function for cluster assignment, partitioning prior to scheduling takes care of the resource load only in an approximate manner. The exact knowledge of load on clusters and functional units is known only while scheduling.

6 Conclusions

We have proposed a new framework for scheduling clustered processors. The framework resolves the phase-ordering and fixed-ordering problems associated with the earlier schemes and provides a mechanism to explore the trade-offs in runtime performance and code size. The framework is generic and requires slight tweaking of heuristics to adapt to a different cluster VLIW configuration and ICC model. We observe about 16% and 28% performance improvement on the average over an earlier integrated scheme and phase-decoupled scheme respectively without code size penalty.

References

[1] MACHINE SUIF. "http://www.eecs.harvard.edu/hube/".

[2] SUIF Compiler System. "http://suif.stanford.edu/".

[3] CPLEX, Using the CPLEX callable library Ver3. "http://www.ilog.com/products/cplex/", 1995.

[4] A. Aleta, J. M. Codina, J. Snchez, and A. Gonzlez. Graph-partitioning based instruction scheduling for clustered processors. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 150–159. IEEE Computer Society, 2001.

[5] F. C. Chow and J. L. Hennessy. The priority-based coloring approach to register allocation. *ACM Trans. Program. Lang. Syst.*, 12(4):501–536, 1990.

[6] G. Desoli. Instruction assignment for clustered VLIW DSP compilers: A new approach. Technical Report, Hewlett-Packard, February 1998.

[7] P. Faraboschi, G. Brown, J. A. Fisher, and G. Desoli. Clustered instruction-level parallel processors. Technical report, Hewlett-Packard, 1998.

[8] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood. Lx: A technology platform for customizable VLIW embedded processing. In *Proceedings of the 27th annual international symposium on Computer architecture*, pages 203–213. ACM Press, 2000.

[9] K. Kailas, A. Agrawala, and K. Ebcioglu. CARS :A new code generation framework for clustered ILP processors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)*, January 2001.

[10] V. S. Lapinskii, M. F. Jacome, and G. A. De Veciana. Cluster assignment for high-performance embedded VLIW processors. *ACM Trans. Des. Autom. Electron. Syst.*, 7(3):430–454, 2002.

[11] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 330–335. IEEE Computer Society, 1997.

[12] R. Leupers. Instruction scheduling for clustered VLIW DSPs. *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (Philadelphia, PA)*, October 2000.

[13] D. Matzke. Will physical scalability sabotage performance gains. *IEEE Computer*, 30(9):37–39, September 1997.

[14] R. Nagpal and Y. N. Srikant. A graph matching based integrated scheduling framework for clustered VLIW processors. Technical Report, Indian Institute of Science, 2004.

[15] R. Nagpal and Y. N. Srikant. Integrated temporal and spatial scheduling for extended operand clustered VLIW processors. In *Proceedings of the first conference on computing frontiers*, pages 457–470. ACM Press, 2004.

[16] E. Ozer, S. Banerjia, and T. M. Conte. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures. In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pages 308–315. IEEE Computer Society Press, 1998.

[17] P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. In *24th ACM/IEEE conference proceedings on Design automation conference*, pages 195–202. ACM Press, 1987.

[18] N. Seshan. High VelociTI Processing. *IEEE Signal Processing Magazine*, March 1998.

[19] Y. N. Srikant and P. Shankar, editors. *The Compiler Design Handbook: Optimizations and Machine Code Generation*. CRC Press, 2002.

[20] A. Terechko, E. L. Thenaff, M. Garg, J. van Eijndhoven, and H. Corporaal. Inter-cluster communication models for clustered VLIW processors. In *Proceedings of Symposium High Performance Computer Architectures*, February 2003.

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