A Graph Matching Based Integrated Scheduling Framework for Clustered VLIW Processors

Rahul Nagpal and Y.N.Srikant

IISc-CSA-TR-2004-13

http://archive.csa.iisc.ernet.in/TR/2004/13/

Computer Science and Automation

Indian Institute of Science, India

October 2004

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

Clustered architecture processors are preferred for consumer electronic devices because centralized register file architectures scale poorly in terms of clock rate, chip area and power consumption. 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 reduce the code size 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. The framework provides a mechanism to exploit the slack of instructions by dynamically varying the cost of scheduling an instruction using different alternatives to reduce the code size and inter-cluster communication without stretching the overall schedule. A better estimate of instruction slack is determined by first scheduling on a unclustered base VLIW. We observe approximately 16% and 28% improvement in the performance over an earlier integrated scheme and a phase-decoupled scheme respectively as well as reduction in code size. We evaluate its effectiveness in improving the runtime performance of the code without code size penalty.

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. ILP architectures have been developed to meet the need for high performance. Two major ILP design philosophies are superscalar architecture and VLIW architecture.

Superscalar processors have dedicated hardware responsible for scheduling instructions at runtime to improve the performance. They also employ a branch predictor to avoid pipeline stalls that occur in the event of control transfer. However the complicated hardware needed to carry out dynamic scheduling leads to complicated design and high power consumption. The memory requirement for branch prediction also contributes to high power consumption, chip area and cost of these architectures. These problems of superscalar processors make them unsuitable for embedded systems. Another design philosophy is the VLIW architecture, where the compiler is responsible for scheduling. This simplifies the hardware but in order to and exploit the ILP in media applications and to satisfy the high performance requirements of these applications more functional units that can operate in parallel are required . This in turn requires more number of read and write ports and hence increased chip area, cycle time and power consumption since for N arithmetic units the area of register files grows as N^3, delay as $N^{3/2}$ and power dissipation as N^3 [17].

Recently clustering has been proposed to overcome these difficulties 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. Inter-cluster communication (ICC) can be provided in various ways such as send-receive model, extended operand model, extended result model and broadcast model. 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 [14] [9].

A compiler for these architecture is responsible for carrying out the complex job of mapping ILP available in an application to the parallelism offered by the underlying hardware. Specifically, a scheduler is responsible for binding operations to resources in different clusters. The problem of scheduling becomes harder in the context of clustered architectures because the scheduler has to decide not only when to schedule (temporal concerns) but also where to schedule (spatial concerns). These decisions are often very complicated because scheduling an instruction at a spatial location affects temporal and spatial scheduling decisions of instructions dependent on this instruction. The scheduler is required to resolve the conflicting goals of exploiting hardware parallelism as well as minimizing communication among clusters, ideally making use of all the facilities provided by hardware without any extra cost.

Different clustered datapath configurations, connectivity types and ICC models proposed [19] in literature have different architectural costs and they present different performance trade-offs to the scheduler in terms of code size and execution time. For example, a partially connected VLIW architecture provides better scalability than a fully connected clustered VLIW architecture but the scheduler is required to work under the constraint of avoiding deadlock due to non-schedulability of certain cross-paths. Similarly, the send-receive ICC model (used in Hewlett-Packard Lx architecture) requires an explicit copy operation for ICC and thus some of the issue slots are occupied by copy operations. This may lead to a delay in scheduling other operations as well as an increase in the code size. A Scheduler for architectures using a send-receive ICC model needs to reduce the communication among clusters to make more free slots available for other instructions without stretching the overall schedule. On the other hand, an extended operand ICC model (used in Texas Instruments VelociTi architecture [18]) attaches a cluster id field with some of the operands and this allows an instruction to read some of operands from register files of other clusters *without any extra delay*. Extended operand ICC model reduces the register pressure as the transferred value is not stored in the local register file but is consumed immediately. However reuse of the transferred value is not possible. A scheduler can leverage the benefits of this model by orchestrating the operations among clusters in such a way that the free ICC facility can be utilized to the maximum possible extent. Minimizing the need for explicit inter-cluster move operations reduces the code size apart from reducing the register pressure and makes available more free slots for other instructions. However due to the architectural cost of providing large no of cross-paths only some of the functional units can read their operands from other clusters and only some of the operands can have fields for specifying the cluster id. These restrictions coupled with limited channel bandwidth add to the severity of efficiently scheduling extended operand clustered processors. The earlier proposals for scheduling clustered VLIW architecture fall into two main categories namely phase-decoupled approach and integrated approach.

Due to the difficulty and complexity of engineering a combined solution there have been many proposals for phase-decoupled scheduling. The phase-decoupled approach of scheduling 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. Other schemes which are geared towards minimizing ICC may reduce ILP to achieve this goal. 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 towards scheduling tries to combat 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 available clusters in some priority order. Priority orders for considering instructions are often based on parameters such as mobility, scheduling alternatives and 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 to reduce communication or to schedule it at the earliest.

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 sub optimal schedule in terms of performance, code size or both. Since capabilities of resources in a realizable clustered VLIW architecture differ in terms of operations they can perform and also number and type of communication they can accommodate, scheduling alternatives for an instruction depend on resource and communication requirements of other instructions ready to be scheduled in the current cycle. We have an experimental evidence that shows that statically decided ordering exhibits different performance characteristics for different DFG even for the same architectural configuration. 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 be easily adapted to accommodate 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. A better estimate of freedom in scheduling an instruction is obtained by first scheduling on an unclustered base VLIW processor with the same configuration. The freedom available in scheduling an instruction and hence the cost of scheduling an instruction using different scheduling alternatives is dynamically varied. Thus the framework provides a mechanism to exploit the slack of instructions to reduce the code size and ICC without affecting the overall schedule length. The proposed algorithm is generic and 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 TI3206401 DSP processor. We evaluate its effectiveness in improving the runtime performance of the code without code size penalty.

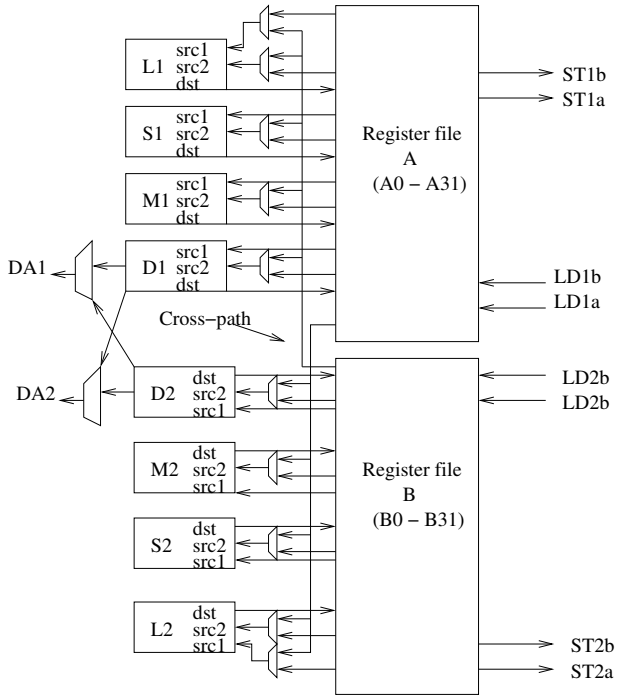

Fig. 1. TI320C64X CPU Data Path

Work is underway to adapt the simulator for a variety of clustered VLIW configurations and performance measurement. We are also in the process of adapting our algorithm for software pipelining of inner loops.

The rest of the paper is organized as follows. Section 2 describes the problem. Section 3 contains our graph matching based scheduling framework and the implementation aspects. Section 4 presents performance statistics and a detailed evaluation based on experimental data. In section 5 we mention related work in the area. We conclude in section 6 with a mention of future work.

2. Problem Description

2.1. Machine Model

We consider a generic machine model having many clusters with the same or different datapath configurations. The connectivity among clusters can be full or partial and the ICC model can be varied. We even consider the possibility of a hybrid communication model where some of the functional units can communicate by reading operands from register file of different clusters (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). Functional units can be varied in terms of 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 have some specialized task delegated to them. This generic machine model enable us to design our framework for a variety of clustered architectures differing

in datapath configuration and communication model. Though we consider a point to point interconnection, framework can be easily adapted for a bus based interconnection. Next we briefly explain the TI320C64X architecture, a production clustered processor having some of the features of our machine model, which we have used for practically evaluation of the framework.

TI320C6401 is a load-store RISC-style architecture. 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 both 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. Six registers (A0,A1,A2,B0,B1,B2) can be used for conditional execution of instructions and all instructions can be executed conditionally. However, some instructions can be conditionally executed by only specific units. The functional capabilities of each unit is as follows:

L unit	Integer addition, logical operations and bit counting
S unit	Integer addition, logical operation, and branch control
M unit	Integer multiplication
D unit	Integer addition and load-store operation

The pipeline is divided into three phases. Fetch phase covers four pipeline stages, Decode phase covers two pipeline stages and Execute phase covers 5 pipeline stages. 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.

2.2. The Scheduling problem

We are given a set of operation types, a set of resource types and a relation between these two sets. This relation may not be strictly a mapping in general since a resource can perform more than one kind of operation and an operation can be performed on more than one kind of resource. There can be more than one instance of each type of resource. Resource instances can be partitioned into sets each one representing a cluster of resources.

Given a data flow graph, which is a partially ordered set (POSET) of operations the problem is to assign each operation a time slot, a cluster and a functional unit in a chosen cluster such that the total number

of time slots needed to perform all the operations in POSET are minimized while the partial order of operations is honored and neither any resource nor the ICC facility is over committed.

Formally we are given

1) A set of operation types *OT*, a set of resource types *RT* and a set of clusters *C*
2) A relation between two sets given by *OR*: *OT*→ *RT* such that $OR(o_i)$ for an operation o_i is a set of resource types $RT_i \subset RT$ that can perform these operations, and
3) A set *L* of triples *(t,c,r)* where *t* is a time slot, $c \in C$ and $r \in R$

A POSET of operations can be represented as a directed acyclic graph G(V,E) where V is a set of operations in the POSET and E represent the set of edges. Edges are labeled with a number representing the time delay needed between the head operation and the tail operation. We use the following notation:

O	Set of operations
o	An individual operations
R	Set of resources
r	An individual resource
N(RT)	No. of instance of resource type *RT*
N(c,RT)	No. of instance of resource type *RT* in cluster *c*
BW(ci,cj)	Communication bandwidth between cluster *i* and cluster *j*
NT(t,ci,cj)	No. of transfer between cluster *i* and cluster *j* in a time step t
RT(r)	The type of resource *r*
OT(o)	The type of operation *o*
C(r)	The cluster of resource *r*
l	An individual triple of *L*
$ES(o_i)$	The subset of edges having o_i as successor

Scheduling in this context is the problem of computing a mapping S from a POSET P to a set of triple L such that the number of time steps needed to carry out all the operations in the POSET is a minimum subject to these constraints (apart from other architectural constraints).

1) $forall\ i$ *such that* $S(o_i) = l \in L$, $l.t \geq max(w(e_j))$ *for all* $j \in ES(o_i)$.
2) $RT(l.r) \in OR(o_i)$
3) $C(l.r)=l.c$
4) $\forall\ t\ \forall\ c\ \forall\ rt \sum (l_i.t = t) \wedge (l_i.c = c) \wedge (RT(l_i.r) = rt) \leq N(c,rt)$
5) $\forall\ t\ \forall i\ \forall j\ NT(t,ci,cj) \leq BW(ci,cj)$

Since the problem of optimal partitioning and resource constrained scheduling of DFG is NP-complete

[8], exponential time complexity renders it impractical to search for an optimal schedule. We solve the problem by considering various scheduling alternatives of instructions ready to be scheduled in each cycle and prioritizing them based on heuristics. The best possible placement of instructions in a particular cycle is determined by formulating the problem as a sequence of two 0-1 integer linear programming problems.

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,u,c)
      end if
      schedule instruction i on unit u in cluster c
    end if
  end while
  readylist.update(G)
  advanceCycle()
end while
```

Fig. 2. Graph Matching based spatial and temporal scheduling algorithm

Our graph matching based scheduler considers the instructions which are ready to be scheduled in each cycle and creates a bipartite graph with nodes as instructions and resources in all clusters. An edge connecting an instruction node and a resource node represents a possible scheduling alternative for the instruction. There are no edges connecting any two instruction nodes and any two resource nodes. There can be more than one edge between an instruction and a resource depending on possible scheduling alternatives for the instruction. Each edge is assigned a cost determined by a cost function and the information regarding usage of cross-paths. The cost of an edge is composed of various factors

that contribute to the priority of scheduling an instruction in the current cycle over other instructions. Contribution of these factors and hence the cost of scheduling an instruction using an alternative is dynamically varied from one cycle to another. We then find a minimum cost feasible matching problem for this graph using an ILP solver. The solution gives the minimum cost feasible scheduling alternatives for the given graph. The second problem is set up such that the space of all feasible matchings having the cost same as that of the minimum cost previously obtained is searched and its solution maximizes the number of instructions in the match. 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. Figure 2 gives a very high level description of our algorithm. 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. A better estimate of instruction freedom is important for efficient binding and exploring the trade-off in code size and performance. For a better estimation, we first schedule on a unclustered base VLIW processor with the same data path configuration but ignoring all communication delays. Scheduling on the base VLIW configuration is carried out using the same matching-based scheduler. The cost function used for scheduling base VLIW is same as one given in figure 4 but does not include any communication cost factor and hence B is set to zero. Since in general the best schedule that can be obtained on clustered VLIW will be of same length as that of base VLIW, these freedom values incorporating resource constraints are used 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. Since dummy edges have infinite cost associated with them and problem is formulated as minimum cost matching problem, the ILP solver mostly tries to ignore them in the final match except when they are indispensable depending upon the resource and communication requirements of instructions under consideration. Thus any dummy edge that is selected in the final match is ignored without affecting the final solution.

Figure 3 gives an outline of the graph construction.

```
input: A readylist of nodes
output: A Matching Graph M
var
ready_list : List of nodes (instructions)
while (!ready_list.is_empty()) do

  for each instruction i in the readylist do

    for each resources r in the all the cluster do

      for each communication and computation alternative a for scheduling instruction on this resource
      do
        c=findCost(i,r,a)
        t=findCommunicationVector(i,r,a)
        add an edge between instruction i and resource r with the cost c and communication vector t

      end for

    end for

  end for
end while
```

Fig. 3. Outline of Graph Construction Algorithm

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. The second ILP problem maximize 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:

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

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

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

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:

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

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

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

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

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.

Table 1 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. The values of different constants can be changed to reflect the ICC model under consideration. Thus the cost function is generic and require slight tweaking of heuristics 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 cost of any of the alternatives

Fig. 4. Cost Function

TABLE 1
VALUES OF CONSTANTS USED IN COST FUNCTION

Constant	Value
α	1.00
β	1.00
X	0.5
Y	0.75
Z	1.00
A	1.00
B	1.00
C	0.10

3.5. Selective Rejection based on Communication and Mobility

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. In order to further reduce the inter-cluster communication and extra code added, 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. Selective rejection thus provides a mechanism to explore the trade-off in code size and performance. We present the performance impact of using and not using selective rejection in terms of execution time and code size in a later section.

3.6. An Example

This subsection presents an example to demonstrate the benefit of using the graph matching based integrating scheduling approach for clustered processors to achieve high performance, low register pressure and reduction in code size. Let us assume that the earlier scheduling decisions were such that the VR2239,VR1260,VR106,VR1512, and VR124 are bound to cluster 1 while VR70, VR90,VR1195,VR60,

TABLE 2

SCHEDULING ALTERNATIVES FOR INSTRUCTIONS

Sr	Instr	Mob	Cons	L1			S1			D1			M1			L2			S2			D2			M2		
				E	S	F	E	S	F	E	S	F	E	S	F	E	S	F	E	S	F	E	S	F	E	S	F
1.1	MVK	4	1				0	0	0													0	0	0			
2.1	MVK	4	1				0	0	1													0	0	0			
3.1	MPY	2	2										1	1	0										0	0	0
3.2													2	0	0												
4.1	MPY	2	2										0	0	0										1	1	0
4.2																									2	0	0
5.1	MPY	2	1																								
5.2																											
6.1	ADD	3	2	0	1	0	0	1	0	0	1	0				0	1	1	0	1	1	0	1	1			
6.2				1	0	0	1	0	0	1	0	0				1	0	1	1	0	1	1	0	1			
7.1	ADD	3	2	0	1	1	0	1	1	0	1	1				0	1	0	0	1	0	0	1	0			
7.2				1	0	1	1	0	1	1	0	1				1	0	0	1	0	0	1	0	0			
8.1	ADD	5	1	0	1	0	0	1	0	0	1	0				0	1	1	0	1	1	0	1	1			
9.1	SUB	4	2	1	0	1	1	0	1							1	0	0	1	0	0						
10.1	SUB	4	2	0	0	0	0	0	0							1	1	1	1	1	1						
10.2																2	0	1	2	0	1						
11.1	SUB	4	1	1	0	1	1	0	1							1	0	0	1	0	0						
11.2				0	1	1	0	1	1							0	1	0									

TABLE 3

COST OF SCHEDULING ALTERNATIVES

Sr. No.	Instr	MF	UF	L1		S1		D1		M1		L2		S2		D2		M2	
				CF	TC	CF	TC	CF	TC	CF	TC	CF	TC	CF	TC	CF	TC	CF	TC
1.1	MVK	0.67	0.03			0	0.7							0	0.7				
2.1	MVK	0.67	0.03			0.27	0.97							0	0.7				
3.1	MPY	0.33	0.0							0.55	0.88							0	0.33
3.2										0.73	1.06								
4.1	MPY	0.33	0.0							0.0	0.33							0.55	0.88
4.2																		0.73	1.06
5.1	MPY	0.33	0.03							0.55	0.91							0	0.37
5.2										0.73	1.09								
6.1	ADD	0.5	0.0	.18	.68	.18	.68	.18	.68			0.45	.95	0.45	.95	0.45	.95		
6.2				.36	.86	.36	.86	.36	.86			0.64	1.14	0.64	1.14	0.64	1.14		
7.1	ADD	0.5	0.0	.45	.95	.45	.95	.45	.95			0.18	.68	0.18	.68	0.18	.68		
7.2				.64	1.14	.64	1.14	.64	1.14			0.36	.86	0.36	.86	0.36	.86		
8.1	ADD	0.83	0.03	.18	1.05	.18	1.05	.18	1.05			0.45	1.32	0.45	1.32	0.45	1.32		
9.1	SUB	0.67	0.0	.64	1.3	.64	1.3					0.36	1.03	0.36	1.03				
10.1	SUB	0.67	0.0	0.0	0.67	0.0	0.67					0.82	1.48	0.82	1.48				
10.2												1.0	1.67	1.0	1.67				
11.1	SUB	0.67	0.03	.64	1.34	.64	1.34					0.36	1.06	0.36	1.06				
11.2				.45	1.15	.45	1.15					0.18	.88						

and VR80 are bound to cluster 2. Figure 5 lists the set of instructions ready to be scheduled in the current cycle. Table 2 enumerates the scheduling alternatives possible for each instruction. Scheduling alternatives for an instruction varies in terms of functional unit and communication options used (explicit move, snooping facility or combination of both). The commutativity of operations like ADD and MPY is taken into account while considering the possible scheduling alternatives. We enumerate all possible scheduling alternatives for all instructions and leave it to the matcher to select among scheduling alternatives with an aim of scheduling as many instructions as possible in each cycle. Table 3 presents the mobility factor (MF),

1. MVK 8, VR109
2. MVK 9, VR147
3. MPY VR70, VR90, VR1231
4. MPY VR2239, VR1260, VR1243
5. MPY VR70, VR60, VR243
6. ADD VR1260, VR1195, VR65
7. ADD VR106, VR70, VR102
8. ADD VR1512, VR60, VR115
9. SUB VR124, VR60, VR112
10.SUB VR124, VR2239, VR118
11.SUB VR1512, VR80, VR181

Fig. 5. Set of Instructions ready to be scheduled

MV .D2X VR106, VR2244(Scheduled in an earlier cycle)

C1.3||MPY .M2 VR70, VR90, VR1231 (3.1)
C1.4||MPY .M1 VR2239, VR1260, VR1243 (4.1)
C1.5||ADD .D1X VR1260, VR1195, VR65 (6.1)
C1.6||ADD .D2X VR2244, VR70, VR102 (7.2)
C1.7||SUB .L2X VR124, VR60, VR112 (9.1)
C1.8||SUB .L1 VR124, VR2239, VR118 (10.1)
C1.1||MVK .S1 8, VR109 (1.1)
C1.2||MVK .S2 9, VR147 (2.1)

Fig. 6. Schedule Generated using GM

C1.1||MV .D1X VR80, VR2246
C1.2||MV .D2X VR124,VR2325

C2.1||MPY .M2 VR70, VR90, VR1231 (3.1)
C2.2||MPY .M1 VR2239, VR1260, VR1243 (4.1)
C2.3||ADD .L1X VR1260, VR1195, VR65 (6.1)
C2.4||ADD .L2X VR106, VR70, VR102 (7.1)
C2.5||SUB .S2X VR2325, VR60, VR112 [1]
C2.6||SUB .S1 VR124, VR2239, VR118 (10.1)
C2.7||ADD .D1X VR1512, VRVR2246, VR181 [1]

Fig. 7. Scheduled Generated using UM

[a]Scheduled using and explicit MV in a separate cycle

uncovering factor (UF), communication cost factor (CF) and total cost (TC) computed for all scheduling alternatives using the cost function described above.

Figure 6 presents the schedule generated by the graph matching based scheduler. Each instruction in the generated schedule is suffixed with the scheduling alternative number used by the scheduler. Graph Matching scheduler tries to pack as many instructions as possible in each cycle while exploiting the communicative and operational capabilities of functional units. It is because all possible scheduling alternatives for all instructions varied in terms of communication and resource usage have been considered simultaneously, graph matching based scheduler is able to achieve maximum possible ILP in this case without generating too many explicit MVs. It requires only one explicit MV operation and that too can be accommodated in a free communication slot available in an earlier cycle. Schedule also tries to assign clusters to instructions in such a way to avoid need for high communication in the future.

The paper by Ozer et el. does not propose any particular priority order for considering instructions. There is also no notion of functional unit binding in the scheme proposed by Ozer et al. However the effective functional unit binding is important in the case of machine model under consideration because

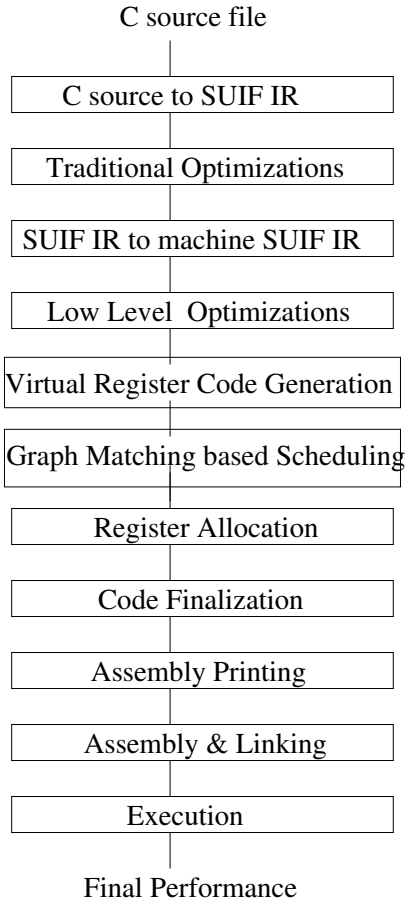

Fig. 8. Scheduling Framework

of variations in the operational and communicative capabilities of functional units. The communication among clusters thus is not fixed but varies depending on functional unit to which an instruction is bound. However a mere cluster assignment mechanism to reduce communication or to perform an instruction at the earliest is unlikely to perform well in general. It may be noted that a trivial extension of UAS in presence of limited snooping facility will prefer to snoop operand where possible to avoid the overheads associated with explicit MV operations. However in the absence of resource and communication requirements of other instructions ready to be scheduled, it may lead to suboptimal schedule. Considering instructions in increasing order of mobility, figure 7 presents a possible schedule generated using UAS algorithm with MWP heuristic for cluster selection. MWP heuristics assign an instruction to cluster where most of the predecessors of the instruction reside. However the MWP heuristics does not take into account the future communication cost of a binding. Instructions 3 and 4 are assigned to cluster 2 and 1. Since there is no conflict in terms of functional unit usage for these instructions, they can be assigned to units M2 and M1 respectively in cluster 2 and 1. Since instructions are considered in increasing order of mobility, the resource conflict due to another MPY instruction is resolved by giving preference to low mobility instructions. Next in priority order are two ADD instructions which have their operands

on alternate clusters. Let us assume that they are assigned to cluster 1 and cluster 2 respectively to reduce the future communication (However MWP heuristics as proposed in the Ozer et al. paper does not consider the future communication cost of a binding and instructions in this case are equally likely to be assigned to any of the cluster). ADD instruction can be scheduled on either of L, S or D unit. Some other instructions ready to be scheduled in current cycle are less flexible in this regard like SUB instruction can be scheduled on L or S unit and MVK operation can be scheduled only on S unit. (Though there is no LOAD instruction ready to be scheduled, on machine model under consideration LOAD instruction is specific about D unit with an additional constraint that two LOAD can be scheduled in a cycle if the addresses are in different clusters and the the value is also loaded in different clusters). Since UAS is not aware of resource and communication requirement of other instructions ready to be scheduled in current cycle, two ADD instructions are scheduled on unit L1 and L2 respectively while snooping one of their operand from other cluster in the schedule shown in figure 7. Next instruction in priority order is a SUB instruction which is equally likely to be assigned to any of the cluster according to MWP heuristics but assigning it to cluster 2 reduces the need for future communication. However none of the cluster have free cross-path to read the operand from other cluster. Even if we assume that the cross-path is available, assigning instruction to cluster 2 requires reading the first operand from the cluster 1. Only L unit has the commutative capability of snooping first operand from other cluster but it is already allocated. Since UAS is a greedy algorithm and does not revisit the earlier scheduling decisions and SUB is not a commutative operation, only way to schedule SUB in current cycle is to schedule an explicit MV operation. Unfortunately the operand VR124 is produced in just last cycle. Thus MV can not be scheduled in any of the earlier cycles despite the availability of cross-path and requisite resource. This require scheduling an explicit MV in a separate cycle and the current cycle becomes cycle 2. The instruction is modified to use VR2325. The next instruction in priority order is again a SUB instruction which is assigned to cluster 1 and can be scheduled on S1. The next two instruction in priority order are two MVK instructions which exclusively demands S unit and neither of them are free. So despite their higher priority they are delayed for consideration in the later cycles. A low priority ADD instruction can be scheduled on available D unit on cluster 1. However it also require an explicit MV due to unavailability of cross-path in current cycle and because the value it requires is produced in just last cycle. The MV is thus scheduled in the cycle 1 and the instructions is modified to use VR2246 in place of VR80. This simple example demonstrates that the schedule generated using an integrated algorithm like UAS suffers from several drawbacks like poor performance, low functional unit utilization and increased register pressure and code size due to more number of explicit MV operations. Some low mobility instructions could not be accommodated in the schedule generated using UAS algorithm and this may lead to further stretch of schedule.

TABLE 4
DETAILS OF THE BENCHMARK PROGRAMS USED

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 5
LEGENDS AND THEIR MEANING

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	Unified Assign and Schedule with Completion Weighted Predecessors Ordering
UM	Unified Assign and Schedule with Magnitude Weighted Predecessors Ordering
LP	Lapinskii pre-partitioning Algorithm

4. EXPERIMENTAL EVALUATION

4.1. Setup

We have used the SUIF compiler [2] along with MACHSUIF library [1] for our experimentation. Figure 8 depicts the flow. We perform most of the optimizations on both high level and low level intermediate code. We generate code for TI TMS320C64X [20]. Ours is a hand-crafted code generator which is designed to take advantage of the specialized addressing modes of the architecture. Code generation is followed by a phase of peephole optimization and this highly optimized code is passed to our scheduler. We have used CPLEX ILP solver [3] to handle the ILP formulation of graph matching. The scheduler annotates 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 TI320C64X architecture [20].

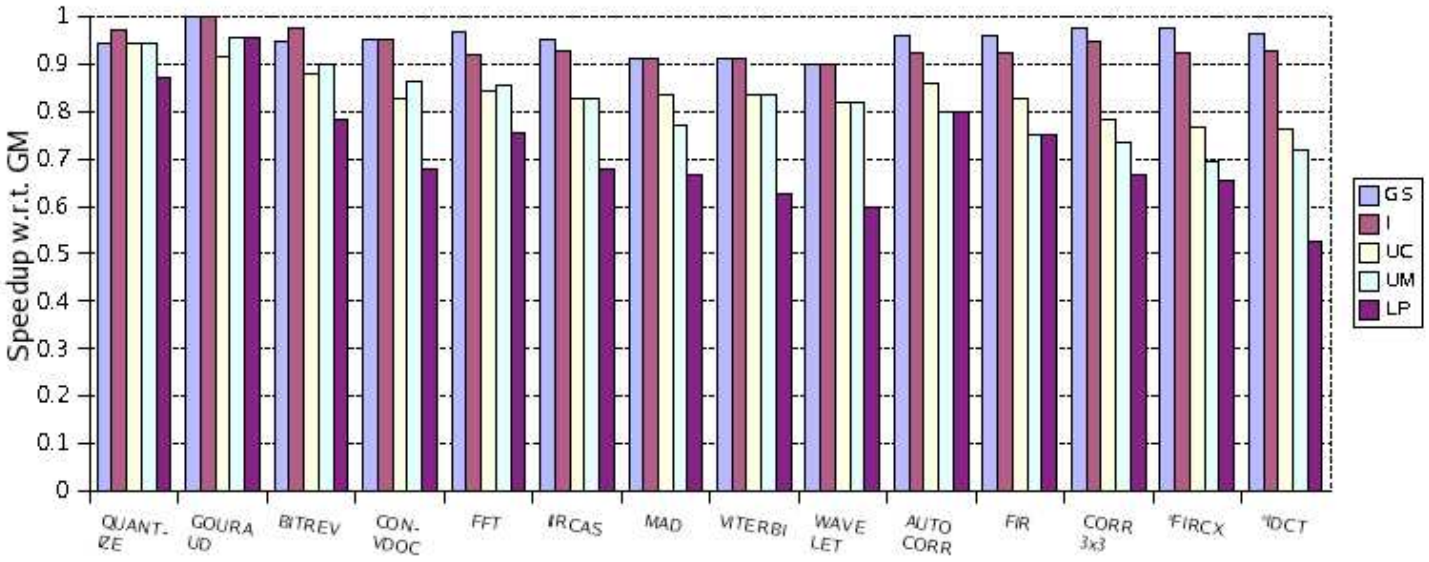

Fig. 9. Speedup as compared of GM

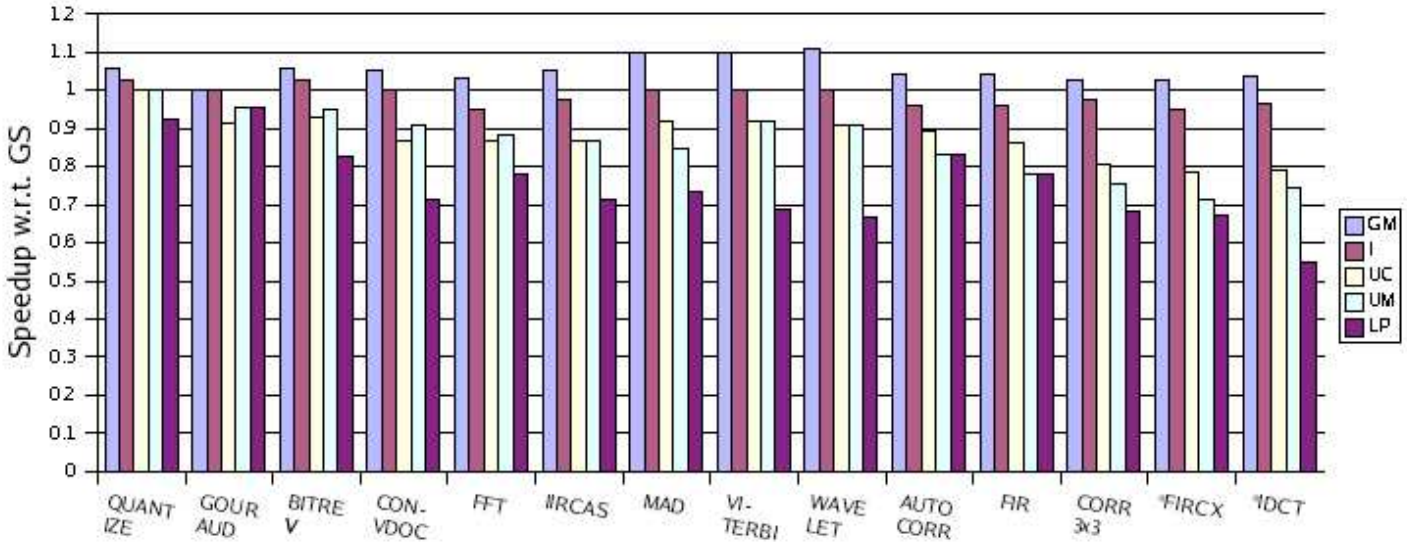

Fig. 10. Speedup as compared to GS

4.2. Performance Statistics

Table 4 summarizes the key characteristics of the benchmark programs we have used for experimental evaluation. These benchmarks are mostly unrolled inner loop kernels of MEDIABENCH [12], a representative benchmark of multimedia and communication applications and is specially designed for embedded applications. Table 4 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 ration 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

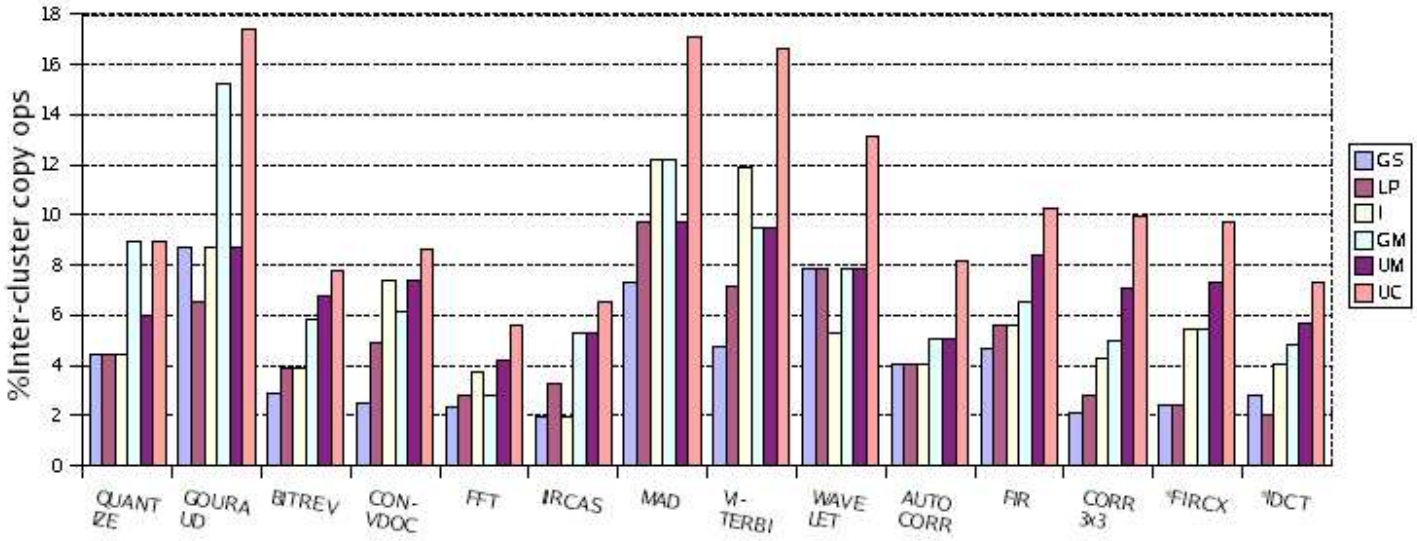

Fig. 11. % Distribution of Explicit Inter-cluster MV Instructions vs. Other Instructions

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. We compare the performance of our algorithm with the unified-assign-and-schedule algorithm(UAS) [15] and pre-partitioning algorithm by Lapinskii et al. [11].

Figures 9 and 10 depict the speed-up of different algorithms as compared to graph matching based scheduling algorithm and graph matching with selective rejection algorithm respectively. 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. [15]. GM attains approximately 6.4%, 16.37%, 18.16%, and 28.51% improvement respectively over I, UC, UM, and LP. GS could attain approximately 1.48%, 11.94%, 13.82%, and 24.87% improvement respectively on the average on I, UC, UM, and LP. GS suffers a slight performance degradation of about 4.97% on the average compared to GM. Figure 11 presents the percentage distribution of explicit inter-cluster MV instructions vs. other instructions for different algorithms. GS and LP perform better then other algorithms in terms of extra code added. GM is better them UM and UC in general and UC performs worse of all. The difference in extra code added by UC becomes more prominent for high ILP routines.

It may be noted that though we formulate and solve the problem as a sequence of two ILP problems, the compilation time is still of the order of milliseconds even for reasonable large DFG. This is because the problem is solved for set of instruction ready to be scheduled in a particular cycle. We believe that even considering more number of clusters and additional communication constraint will not increase the compilation time to an extent that may question the practicability of approach given the quality of

generated schedule and the importance of quality code in the embedded domain.

4.3. Discussion

In this section we discuss the performance of GM and our implementation of the UAS algorithm [15] and Lapinskii pre-partitioning algorithm [11]. The Lapinskii pre-partitioning algorithm suffers from well known phase-ordering problem. We consider a fully pipelined machine model where each resource can initiate an operation every cycle. Resources varies in terms of their operational and commutative capabilities. Since on such a model the resource and communication constraints are precisely known only while scheduling, any load balancing based on approximate schedule length leads to suboptimal schedule. Though UAS integrates cluster assignment into the list scheduling algorithm and improves over phase decoupled scheduling approaches, it suffers from the fixed-ordering problem. Ozer et al. have proposed many orders for considering clusters (their paper does not mention any specific order for considering instructions). On the other hand, GM considers all possible scheduling alternatives varied in terms of communication options, spatial locations and resource usage simultaneously instead of following a fixed order. The scheduler simultaneously selects the alternatives for instructions to be scheduled in current cycle while exploiting the communication facility and parallelism offered by hardware. A Cost function composed of various dynamically varying factors such as communication cost, freedom available in scheduling the instruction and uncovering factor of the instruction is used to select from scheduling alternatives for instructions competing for limited resources while scheduling maximum number of instructions in each cycle. Instructions with lesser freedom are scheduled in preference to those with higher freedom values despite their high communication cost, to avoid stretching the overall schedule. Scheduling alternatives for instructions with enough freedom and high communication cost are assigned a high cost to prefer scheduling instructions with low freedom values in the current cycle and in the hope of scheduling high overhead instructions in later cycles using a low overhead alternative.

5. PREVIOUS WORK

Recently there have been several proposals for scheduling on clustered VLIW processors. As mentioned earlier there are two main approaches to scheduling on clustered VLIW architecture. Integrated spatial and temporal scheduling and spatial scheduling followed by temporal scheduling. Work in the first direction is described in Ozer [15], Leupers [13], and Kailas [10] while the phase decoupled schemes are due to Gonzalez [4], Desoli [6] and Lapinskii [11]. In what follows we describe these approaches briefly.

Ozer et al. [15] have proposed an algorithm called unified-assign-and-schedule (UAS) that perform the combined job of partitioning and scheduling. 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. The result show significant performance improvement over the bottom up greedy (BUG) algorithm on benchmark programs selected from SPECINT95 and MEDIABENCH. The authors conclude that UAS with CWP heuristics yields the best performance. 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.

Leupers [13] proposed an algorithm for integrated scheduling of clustered VLIW processors. They use simulated annealing, a heuristic search algorithm for cluster binding followed by a list scheduling algorithm.

Kailas et al. [10] have proposed an algorithm for combined binding, scheduling and register allocation for clustered VLIW processors. Their greedy algorithm binds a node to the cluster in which it can be executed at the earliest. This may lead to a high inter-cluster communication in the future (while scheduling successor of this node) due to the unavailability of a communication slot to schedule the required MV operation and this may stretch the schedule as well. They also propose an on-the-fly register allocation scheme. Insertion of spill code and scheduling of spill code is also integrated in the main algorithm.

In general the integrated schemes proposed earlier are too specific about a particular ICC model and mostly target architectures with explicit MV operations. Though these schemes can be modified for other ICC models, we are not aware of any work in this direction. These schemes also suffer from the fixed ordering problem as mentioned earlier. Simultaneous consideration of various scheduling alternatives is important to avoid unnecessary constraints on scheduling those instructions which are considered late. Targeting any of these schemes for a realizable clustered architecture requires major variations to achieve optimal performance.

Next we discuss some recent proposals for phase decoupled scheduling for clustered processors and contrast these schemes with ours.

Lapinskii et al. [11] 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 with the following components

1) As late as possible scheduling time of an instruction
2) Mobility of an instruction
3) Number of successors of an instruction

They compute the cost of allocating an instruction to a cluster using a cost function given as

$$icost(v,c) = fucost(v,c) * dii(v) * \alpha + buscost(v,c) * dii(BUS) * \beta + trcost(v,c) * lat(BUS) * \gamma \quad (1)$$

icost(v,c)	The cost of binding the instruction v to cluster c
fucost(v,c)	The functional unit serialization penalty
buscost(v,c)	The bus serialization penalty
trcost(v,c)	The data transfer penalty of scheduling v on cluster c.
dii(v)	The data introduction interval of the resource on which v is scheduled
dii(BUS)	The data introduction interval of the BUS
lat(BUS)	The latency of the BUS

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 [16] 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, as we have mentioned earlier, partitioning prior to scheduling takes care of the resource load in only an approximate manner. The exact knowledge of load on clusters and functional units is known only while scheduling.

Gonzalez et al. [4] have proposed a graph partitioning based approach that is used as a pre-partitioning phase to modulo scheduling. In their approach a multi level graph partitioning approach is used to bind the instructions of a DFG to clusters. The first phase is graph coarsening in which a maximum edge weight matching algorithm is used to find a graph matching and collapsing the match nodes into one. This is repeated until the number of instructions are same as the number of clusters. This is followed by

the partition refining phase that considers possible movement of instructions among clusters with a view to get better execution time.

This work is geared more towards finding a partition for cyclic regions so as to reduce the initiation interval and get a better modulo schedule, whereas we concentrate on acyclic graphs.

Desoli and Faraboschi [6] have proposed an elegant algorithm for spatial scheduling called partial component clustering (PCC). The algorithm has two phases. In the first phase the DAG to be scheduled is partitioned into several components using a depth-first search algorithm and a maximum component size threshold. These components are then clusterized while striving for minimum ICC and balancing load among clusters. The second phase is characterized by improving the initial binding of instructions using an iterative algorithm to reduce the schedule length or ICC. The major objective of this algorithm is to minimize the communication among clusters.

This algorithm will perhaps perform well for clustered architectures where the communication among clusters is possible only through separate MV instructions. In the case of extended operand clustered VLIW architectures mere communication reduction may not produce the optimal schedule because the emphasis should be on exploiting free communication slots to map the application ILP to hardware.

6. Conclusions and Future directions

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 as well as reduction in code size.

We propose the following extensions to this work

1) Performance measurement for various clustered VLIW configurations. We are in the process of adapting a public domain simulator for various clustered configurations and ICC models.
2) Adapting the algorithm for software pipelining of inner loops.

7. Acknowledgments

We wish to express our gratitude to Glenn Holloway of MACHSUIF team for prompt replies to our queries during the implementation of our scheduling framework.

References

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

[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] M. R. Gary and D. S. Johnson. *Computers and Intractability - A Guide to the theory of NP-Completeness*. Freeman, 1979.

[9] R. Ho, K. Mai, and M. Horowitz. The Future of Wires. *Proceeding of IEEE*, April 2001.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens. Register organization for media processing. In *Proceedings of the Sixth International Symposium on High Performance Computer Architecture*, pages 375–386, 2000.

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

[19] 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.

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