SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2008; **38**:227–257
Published online 29 May 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.826

Pragmatic integrated scheduling for clustered VLIW architectures

Rahul Nagpal and Y. N. Srikant*,†

Department of CSA, Indian Institute of Science, Bangalore 560012, India

SUMMARY

Clustered architecture processors are preferred for embedded systems because centralized register file architectures scale poorly in terms of clock rate, chip area, and power consumption. Scheduling for clustered architectures involves spatial concerns (where to schedule) as well as temporal concerns (when to schedule). 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. In this paper, we describe our experience with developing a pragmatic scheme and also a generic graph-matching-based framework for cluster scheduling based on a generic and realistic clustered machine model. The proposed scheme effectively utilizes the exact knowledge of available communication slots, functional units, and load on different clusters as well as future resource and communication requirements known only at schedule time. The proposed graph-matching-based framework for cluster scheduling resolves the *phase-ordering* and *fixed-ordering problem* associated with earlier schemes for scheduling clustered VLIW architectures. The experimental evaluation in the context of a state-of-art commercial clustered architecture (using real-world benchmark programs) reveals a significant performance improvement over the earlier proposals, which were mostly evaluated using compiled simulation of hypothetical clustered architectures. Our results clearly highlight the importance of considering the peculiarities of commercial clustered architectures and the hard-nosed performance measurement. Copyright © 2007 John Wiley & Sons, Ltd.

Received 10 July 2006; Revised 2 March 2007; Accepted 3 March 2007

KEY WORDS: scheduling; clustered VLIW architectures; cluster scheduling

*Correspondence to: Y. N. Srikant, Department of CSA, Indian Institute of Science, Bangalore 560012, India.
†E-mail: srikant@csa.iisc.ernet.in

1. INTRODUCTION

The 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 system. Apart from very high performance they also demand low power consumption, low cost, and less chip area to be practical.

Instruction-level-parallel architectures have been developed to meet the high-performance need. These architectures exploit the abundant fine-grained instruction-level-parallelism (ILP) [1] available in embedded applications to satisfy their high performance requirements. Superscalar architectures [2] and very long instruction word (VLIW) [3] architectures are two traditional ILP design philosophies [1]. Both superscalar and VLIW processors have multiple pipelined functional units, which are connected to a single unified register file in parallel to attain better performance. A superscalar processor [2] uses dedicated hardware for scheduling instructions at run time and employs a branch predictor to avoid pipeline stalls that occur in the event of control transfer. However, it suffers from problems such as complicated design, large chip-area, and high power consumption attributed to complicated runtime scheduling logic [4].

A VLIW architecture [3] gets rid of scheduling hardware and associated problems by exposing instruction latencies and delegating the task of scheduling to a compiler. High operation rates as required by emerging real-time embedded applications can be attained by increasing the number of parallel functional units in a VLIW architecture. However, as the number of arithmetic units in a processor increases to higher levels, register storage and communication between arithmetic units become critical factors dominating the area, cycle time, and power dissipation of the processor. The cycle time is determined by the sum of wire delay and gate delay along the critical execution paths of a processor pipeline. Wire delays have become significant for the 0.25 micrometer CMOS process generation and centralized monolithic architectures (both superscalar and VLIW), which use long wires for connecting spatially separated resources may not benefit from the advancements in semiconductor technology [5,6].

A clustered VLIW architecture [7–9] has been proposed to overcome the difficulties with centralized architectures and to make them suitable for use in embedded systems. A clustered VLIW architecture [9] 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 interconnected using some interconnection topology and communication can be enabled using any of the various inter-cluster communication models [10]. Clustering avoids area and power consumption problems of centralized register file architectures while retaining high clock speed, which can be leveraged to obtain better performance. Texas Instrument's VelociTI [11], HP/ST's Lx [12], Analog's TigerSHARC [13], and BOPS' ManArray [14] are examples of the recent commercial clustered micro-architectures.

Although clustering enables higher clock speed, it also incurs overheads in terms of execution cycles and code size. The inter-cluster communication delays are governed by spatial distance between communicating clusters, as well as the latency and the bandwidth of the interconnection mechanism [10]. High quality partitioning/scheduling of computation among clusters is of paramount importance in order to reduce the clustering overheads and to achieve better performance on such an architecture. The problem of scheduling becomes harder in the context of clustered architectures because a scheduler is required to decide not only when to schedule (temporal concern) but also where to schedule (spatial concern) because scheduling an instruction at a spatial location affects

DOI: 10.1002/spe

temporal and spatial scheduling decisions of instructions dependent on this instruction. Furthermore, different clustered datapath configurations, connectivity types, and inter-cluster communication (ICC) models [10] that have been proposed exacerbate the cluster scheduling problem by presenting different performance trade-offs to the scheduler in terms of code size and execution time. Essentially, effective scheduling on such an architecture demands meeting the conflicting goals of exploiting hardware parallelism as well as minimizing communication among clusters.

In this paper, we propose a pragmatic scheme and a generic graph-matching-based framework for integrated spatial and temporal scheduling of clustered architectures [15]. We target a generic and realistic clustered machine model (which is derived from commercial clustered architectures) than the model used in earlier proposals. An experimental evaluation of the proposed framework and some of the earlier proposals is presented in context of a state-of-art commercial clustered architecture. More specifically, the major contributions of this paper can be stated as follows.

- We propose a pragmatic scheme for the integrated scheduling of clustered architectures. The proposed scheme effectively utilizes the exact knowledge of available communication slots, functional units, and load on different clusters as well as future resource and communication requirements known only at schedule time to attain approximately 24% and 11% performance improvement without experiencing a code size penalty over an earlier phase-decoupled [16] and phase-coupled [17] approach to scheduling respectively. This is a purely heuristic based, fast, and low-overhead algorithm.
- We propose a generic graph-matching-based framework (developed on the basis of insight gained during the development of the integrated scheme) for scheduling clustered architectures that resolves the *phase-ordering* and *fixed-ordering problem* associated with scheduling clustered VLIW architectures by simultaneously considering various temporal and spatial scheduling alternatives of instructions. Our graph-matching-based scheduler attains approximately 28.5% and 16% performance improvement over an earlier phase-decoupled [16] and phase-coupled [17] approach to scheduling, respectively. The proposed framework is generic in terms of the number of clusters and numbers and types of functional units in each cluster and requires only slight tweaking of heuristics to get optimal performance for different clustered VLIW configurations and ICC models. The overheads of the graph-matching-based framework are comparatively higher (but not impractical) than the integrated scheme.
- We have implemented the above two proposals and some of the earlier proposed algorithms for scheduling clustered architectures on the Texas instruments' VelociTI architecture [11], a state-of-art production clustered architecture, using SUIF/MACHSUIF [18,19] compiler framework. We present a detailed performance analysis based on an experimental evaluation of these algorithms. To the best of our knowledge, this is the first step towards hard-nosed evaluation of different approaches to scheduling on clustered architectures.

The rest of the paper is organized as follows. In Section 2 we describe the problem of cluster scheduling as well as our machine model, and in Section 3 we present related work in the area and a description of how our work contrasts with the related work, and in Section 4 we describe our integrated scheduling algorithm [15]. In Section 5 we present our graph-matching-based scheduling framework, and in Section 6 we describe our experimental setup, performance results, and a detailed comparative evaluation of proposed and some of the earlier algorithms based on performance results. We conclude in Section 7 with some directions for the future extensions of this work.

DOI: 10.1002/spe

2. THE PROBLEM DESCRIPTION

The problem of scheduling becomes harder in the context of clustered architectures because a scheduler has to decide not only when to schedule (temporal concern) but also where to schedule (spatial concern). Moreover, scheduling an instruction at a spatial location affects temporal and spatial scheduling decisions of instructions dependent on this instruction. In general, the wrong spatial assignment has a much higher cost than the wrong temporal assignment. This is because of the recurring cost of communication and resource contentions each time any dependent instruction is scheduled [20]. Thus a high-quality partitioning/scheduling of computation among clusters is of paramount importance in order to inhibit the clustering overheads from compensating the benefits of better clock speed on a clustered architecture.

Different clustered datapath configurations, connectivity types, and ICC models [10] exacerbate the cluster scheduling problem by presenting different performance trade-offs to the scheduler in terms of code size and execution time. For example, the send–receive ICC model [10] (used in Hewlett-Packard Lx architecture [12]) requires an explicit copy operation for ICC and thus some of the issue slots are occupied by a ICC copy operation and this may lead to a delay in the scheduling of other operations as well as increase in overall code size. A scheduler for an architecture supporting a send–receive ICC model needs to reduce the communication among clusters without stretching the overall schedule length in order to reduce code size and to make available more free slots for other instructions. On the other hand, an implementation of the extended operand ICC model (used in Texas Instruments VelociTI architecture [21]) 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*. The 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 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 extent. Minimizing the need for any explicit inter-cluster move operation reduces the code size apart from reducing the register pressure and makes available more free slots for other instructions. 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 clustered processors. In addition, some instructions on clustered architectures can be performed only on some specialized resources owing to performance or correctness concerns. 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.

2.1. The machine model

We target a generic clustered machine model based on recently developed commercial clustered architectures. In our machine model a clusters can be homogeneous having identical functional units and register file (as with the VelociTI architecture [21]) or heterogeneous with different cluster having a different mix of functional units and register file (as with the HP Lx architecture [12]). The connectivity among clusters can be full or partial. The functional units can vary in terms of the their operational and communicative capabilities. Functional units may be made of operational subunits each of which

DOI: 10.1002/spe

can execute a different kind of operation and there may be more than one instance of each kind of functional unit. Hence an operation can be performed on more than one type of resource and a resource can perform more than one kind of operation. 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 [21]), while communication among some of the clusters is possible only through an explicit move (MV) operation (as in HP Lx architecture [12]). Explicit MV operation takes extra cycles to transfer a value from one register file to another register file whereas the snooping capability allows a limited number of operands to be read from the register files of another cluster in the same cycle. The major implementation cost of providing a snooping capability is the increase in the clock period. So, the number of cross-cluster reads is restricted to say one. The 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 particular resources owing to performance or correctness concerns. This machine model enables us to design a generic and pragmatic framework that can accommodate architecture-specific constraints and can be easily adapted to a variety of clustered architectures differing in datapath configurations and/or communication models. A detailed description of commercial clustered architectures on which we base our machine model is available in the associated technical report of Nagpal and Srikant [15].

Formally, the problem of cluster scheduling can be described as follows. 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.

3. RELATED WORK

Earlier proposals for scheduling on clustered VLIW architectures can be classified into two main categories, namely the phase-decoupled approach and the phase-coupled approach. We briefly mention the earlier work done in both the directions and describe two approaches in each category against which we directly compare our algorithm. We refer the reader to the associated technical report by Nagpal and Srikant for a detailed description [15].

The phase-decoupled approach to scheduling works on a data flow graph (DFG) and performs the partitioning of instructions into clusters to reduce ICC while approximately balancing the load among clusters. The annotated DFG is then scheduled using a traditional list scheduler while adhering to earlier spatial decisions. This approach is known to suffer from phase-ordering problem.

Lapinskii *et al*. [16] have proposed an effective binding algorithm for clustered VLIW processors. Their algorithm performs the spatial scheduling of instructions among clusters and relies on a list

DOI: 10.1002/spe

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 instruction;
2. mobility of instruction;
3. number of successors of instruction.

They compute the cost of allocating an instruction to a cluster using a cost function that takes into account the load on the resources and buses as well as the cost of ICC. Although they have proposed a good cost function for cluster assignment, partitioning prior to the scheduling phase takes care of the resource load only in an approximate manner. The exact knowledge of the load on clusters and functional units is known only while scheduling. This inexact knowledge may lead to suboptimal schedules.

An integrated approach to 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 the available clusters in some priority order. The priority order for considering instructions is decided based on mobility, scheduling alternatives, the number of successors of an instruction, etc. Similarly, the priority order for considering clusters is decided based on the communication cost of assignment, earliest possible schedule time, etc. An instruction is assigned a cluster to reduce communication or to schedule it at the earliest. The proposals in this direction are due to Ozer *et al*. [17], Leupers [22], Kailas [23], and Zalamea [24].

Ozer *et al*. [17] have proposed an algorithm called unified-assign-and-schedule (UAS) that does the combined job of partitioning and scheduling. They extend the list scheduling algorithm with cluster assignment. After picking the highest priority node, a priority list of clusters is formed using some priority heuristic. Each cluster in the prioritized list of clusters is examined to determine whether the ready operation can be scheduled on it. After an available cluster is found, the algorithm checks if any copy operations are required from the other clusters. If the copies can be scheduled on their respective clusters in the previous cycles, the current operation and associated copies are scheduled. A copy operation is needed when a flow dependent predecessor of the operation being scheduled is scheduled on a different cluster. In such cases, the copy operation is attempted to be scheduled on the same cluster with the predecessors operation in a cycle time with an interval of bus latency. They proposed various ways of ordering clusters for consideration as follows:

1. *no ordering* – the cluster list is not ordered;
2. *random ordering* – the cluster list is ordered randomly;
3. *magnitude weighted predecessors (MWP)* – an operation can be placed into a cluster where the majority of the input operands reside;
4. *completion weighted predecessors (CWP)* – this gives higher priority to those clusters that will be producing source operands for the operation, late in the schedule.

3.1. Our proposals

Earlier proposals for scheduling on clustered architectures are specific about a particular ICC model and mostly target architectures that support ICC using an explicit MV operation. However, modern commercial clustered processors also provide limited snooping capabilities. The restrictions related to

DOI: 10.1002/spe

snooping capabilities add to the severity of efficiently scheduling on modern clustered architectures and these have not been addressed by the earlier proposals. More specifically, our work differs from the earlier work in the area in the following ways.

- We target a more generic and realistic machine model (derived from commercial clustered architectures) than the one used by the earlier proposals.
- We perform an effective functional unit binding that we found to be very important for the generic and realistic machine models under consideration where resource and communication constraints are tightly integrated and thus scheduling alternatives of an instruction vary depending on the resource and communication requirements of other instructions ready to be scheduled in the current cycle. The proposed graph-matching-based scheduler finds the best binding of a set of instructions ready to be scheduled in the current cycle among all possible scheduling alternatives. Thus it resolves the fixed-ordering problem associated with earlier integrated algorithms.
- Our scheduling algorithms take into account any future communication that may arise as the result of a binding to resolve the common problem associated with earlier integrated scheduling proposals that greedily reduce communication in the current cycle without considering the future communication cost of a binding.
- Our experimental results are based on running scheduled and register-allocated code for a commercial clustered architecture rather than a compiled simulation of a hypothetical clustered architecture as is the case with most of the earlier proposals.

4. INTEGRATED TEMPORAL AND SPATIAL SCHEDULING ALGORITHM

In this section, we describe our pragmatic scheme for integrated scheduling of clustered architectures [15]. The proposed scheme effectively utilizes the exact knowledge of available communication slots, functional units, and load on different clusters as well as future resource and communication requirements known only at schedule time to attain approximately 24% and 11% performance improvement without extra code size penalty over an earlier phase-decoupled [16] and phase-coupled [17] approach to scheduling, respectively.

4.1. The algorithm

Our algorithm extends the standard list scheduling algorithm and includes following steps:

1. prioritizing the ready instructions;
2. assignment of a cluster to a selected instruction;
3. assignment of a functional unit in the chosen cluster to a selected instruction.

Figure 1 gives a very high-level description of our algorithm. The algorithm consists of the steps mentioned above inside two loops. The first loop continues the scheduling attempts until all operations are scheduled, by incrementing the cycle after each iteration. The second loop ensures that each operation in the ready list is tried at least once in each cycle. In what follows, each of these steps are described in detail in separate subsections.

DOI: 10.1002/spe

```
input: A dataflow graph
output: A triple (slot,cluster,unit) for each instr.
var
ready_list : priority queue of instr.
i : instruction under consideration
cluster_list : priority queue of clusters
c : cluster under consideration
fu_list : priority queue of functional units
u : functional unit under consideration
ready_list.init(G)
while (!ready_list.is_empty()) do
  while (!readylist.allTried()) do
    i ⇐ ready_list.dequeue ()
    cluster_list.init (i)
    c ⇐ cluster_list.dequeue()
    fu_list.init(i,c)
    u ⇐ fulist.dequeue()
    schedule (i, c, u)
    if explicitMvNeeded(i, c, u)) then
      scheduleExplicitMv(i,c,u)
    end if
    mark i as scheduled on unit u in cluster c
  end while
  readylist.update(G)
  advanceCycle()
end while
```

Figure 1. Integrated algorithm.

4.1.1. Prioritizing the nodes

Nodes in the ready list are ordered by a three-component ordering function. The three components of ordering function are as follows:

1. *vertical mobility* – which is defined as the difference between the latest start time (LST) and the earliest start time (EST) to schedule an instruction;
2. *horizontal mobility* – which is defined as the number of different functional units capable of executing the instruction;
3. *number of consumers* – which is defined as the number of successor instructions dependent on the instruction in the DFG.

In a VLIW architecture, the horizontal mobility of an instruction is indicative of the freedom available in scheduling an instruction in a particular cycle. An instruction which is specific about a particular functional unit should be given priority over one with many possible scheduling alternatives.

DOI: 10.1002/spe

current_comm	Number of snoops in the current cycle
future_comm	Number of future communications required
explicit_mv	Number of MV in the earlier cycles

Figure 2. Various forms of communications.

The rationale behind using this three-component ordering function is as follows. The instructions with less vertical mobility should be scheduled early and are given priority over instruction with more vertical mobility to avoid unnecessary stretching of the schedule. Instructions with the same vertical mobility are further ordered in decreasing order of their horizontal mobility. An instruction with a greater number of successors is more constrained in the sense that its spatial and temporal placement affects the scheduling of more number of instructions and hence should be given priority.

4.1.2. Cluster assignment

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

- the chosen cluster should have at least one free resource of the type needed to perform this operation;
- it should be possible to satisfy the communication needs for the operands of this instruction on the cluster either by snooping the operands or by scheduling explicit MV instructions in the earlier cycles given the bandwidth of the channels among clusters and their usage.

Selection of a cluster from the set the feasible clusters is done primarily based on the following communication cost metric:

$$comm\_cost = A * current\_comm + B * future\_comm + C * explicit\_mv \quad (1)$$

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 the combination of different ICC models. The communication cost is computed by determining the number and type of communication needed by a binding (refer Figure 2). *explicit_mv* takes care of communications that can be due to non-availability of snooping facility in a particular cycle (because of cross-path saturation) or in the architecture itself. Only those clusters which have enough communication slots and required resources for scheduling MV in the earlier cycles are considered. *current_comm* is determined by number of operands that reside on a cluster other than the cluster under consideration and can be snooped using the snooping facility available in current cycle. *future_comm* is predicted by determining the successors of this instruction which have one of their parents bound on a cluster different from the current one. In the case some of the parents of some successor are not yet bound, the calculation is done assuming that it can be bound to any of the clusters with equal probability.

DOI: 10.1002/spe

The selection of A, B, and C is architecture specific and depends on available communication options in a clustered architecture and their relative cost. We found that $A = 0.5$, $B = 0.75$, and $C = 1.0$ work well in practice for an architecture that has a limited snooping facility available such as Texas Instrument's VelociTI [11]. In general, the value of A, B, and C are tuned based on target architecture. Since *explicit_mv* is a pure overhead in terms of code size and register pressure, it is assigned double the cost of the *current_comm* to discourage these explicit MVs and favors snooping possibility in the cost model proposed. *future_comm* is also assigned a smaller 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. In case of a tie in the communication cost, the instruction is assigned to a cluster where it can be scheduled on a unit least needed in the current cycle with the same communication cost (this is discussed in greater detail in the following section).

4.1.3. Functional unit binding

Before scheduling for a particular cycle begins, the ready list is preprocessed to generate the resource need vector, which contains the resource requirements of the instructions currently in the ready list. This vector is updated with any changes in the ready list and is used as a metric for effective functional unit binding. We select a functional unit which is least needed in the current cycle, considering all the instructions in the ready list, and assign this unit to the instruction under consideration. This avoids the situation where an instruction can not be scheduled in the current cycle because a specialized resource needed by an operation later in the ready list is not available as the result of a naive decision taken earlier. Since the snooping capability can be limited and only some of the resources may have capability to snoop both the operands from other clusters, the commutativity of operations such as ADD and MPY is exploited to preserve such a resource as far as possible for prospective non-commutative instruction.

Instructions with the same vertical mobility value but different horizontal mobility values are presented in increasing order of horizontal mobility by the ordering function to help the binding of as many instructions as possible in the current cycle. Since some instructions may have low vertical mobility, they are presented earlier for scheduling consideration despite their high horizontal mobility. Owing to limitations on operand snooping capabilities of functional units, an instruction may require the snooping of a particular operand while none of the free functional units available has capability to snoop the desired operand. Our algorithm handles such situations by shuffling functional units already bound to instructions in the current cycle and free units available. If possible we bind an instruction (which is already scheduled) to alternative units and make a unit free with the desired capability for the instruction under consideration.

It may be noted that a blind allocation of functional units may lead to stretched schedules. A blind allocator may allocate an operation to a unit with some special capability while it could have been scheduled on an alternate free unit.

4.2. Scheduler description

Initially the DFG is preprocessed and various DAG properties such as EST, LST, and mobility of each of the node are computed. This is followed by the initialization of the ready list with

DOI: 10.1002/spe

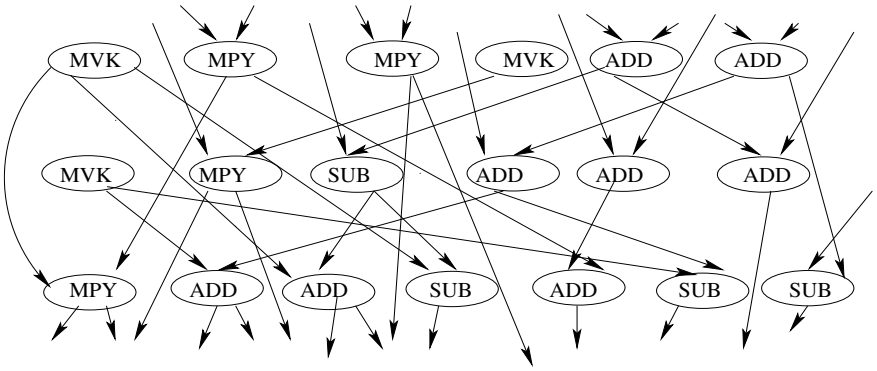

Figure 3. Partial DFG of a media application.

the root nodes of the graph. The ready list also has an associated resource vector which is used to keep track of the approximate resource requirements of the nodes currently in the ready list. The enqueue method of ready list imposes a priority on nodes using a three-component priority function described above. The dequeue method simply returns the next untried node (operation) from the priority ordered ready list and this node is then used to initialize a priority list of clusters on which an instruction can be scheduled in the current cycle. The dequeue method of prioritized cluster list returns the highest priority cluster. The selected cluster and the node are then used to select feasible functional units on the cluster for scheduling the instruction under consideration. The dequeue method of the functional unit list returns the least-needed resource in current cycle for binding the instruction. Any explicit MV operation if needed in an earlier cycle is now scheduled. The cluster initialization routine guarantees that the free slot is needed for an explicit MV operation, it is available before putting the cluster in the feasible cluster list. If an operation can not be scheduled on any cluster the operation is marked as tried and the next operation is considered. Once all the operations have been considered the cycle counter is advanced and ready list and the resource need vector are updated with operations which are now ready for scheduling.

4.3. An example

In this section we present an example to demonstrate the benefit of integrating scheduling decisions for extended operand clustered processors to achieve high performance without experiencing a severe code size penalty. Figure 3 is a partial DFG typical of a DSP application. Figures 4–6 present the schedule generated by our integrated scheduler, a possible schedule using UAS [17], and schedule possible using the pre-partitioning scheme [16], respectively.

We consider the TMS320C6X architecture. The maximum number of instructions that can be executed in a cycle is eight and the hardware provides one communication in each direction per cycle without any extra cost. Assume that the earlier scheduling decisions were such that instructions

DOI: 10.1002/spe

```
1.1MVK .S1 8,V109
1.2||MPY .M2 V70,V90,V1231
1.3||MPY .M1 V2239,V1260,V1243
1.4||MVK .S2 9,V147
1.5||ADD .L1X V1260,V1195,V65
1.6||ADD .D2X V106,V70,V102

2.1MVK .S2 12,V171
2.2||MPY .M2 V1254,V147,V1255
2.3||SUB .L1 V65,V67,V108
2.4||ADD .D2 V102,V103, V107
2.5||ADD .S1X V1230,V1254,V89
2.6||ADD .D1 V1242,V65,V95

3.1MPY .M2X V1231,V109,V1273
3.2||ADD .D2 V107,V171,V114
3.3|| ADD .D1 V109,V108,V115
3.4|| SUB .L1 V109,V108,V116
3.5|| ADD .S1X V89,V1231,V100
3.6|| SUB .S2 V107,V171,V119
3.7|| SUB .L2 V103,V102,V118
```

Figure 4. Integrated schedule.

producing V2239, V1260, V106, V67, V1242, and V1230 are bound to cluster 1 while instructions producing V70, V90, V1195, V103, and V1254 are bound to cluster 2.

Our scheme tries to schedule as many instructions as possible in the current cycle while minimizing the need for communication in the current as well as future cycles. Wherever possible, free-of-cost communication is used to exploit the available ILP. Commutativity of operations such as ADD and MPY are exploited wherever possible to combat the limitations in the snooping capability of the functional units. The ordering function proposed together with the systematic functional unit binding mechanism enable the packing of as many instructions as possible in each cycle.

The pre-partitioning scheme [16] tries to assign clusters in such a way as to reduce the communication while trying to balance the load among clusters. Finally, our temporal scheduler is used to map functional units to the instructions in the cluster assigned by the partitioner. In our implementation of the pre-partitioning scheme we prefer to snoop operands wherever possible to assure fairness in comparison with our integrated approach.

The UAS algorithm with MWP heuristic for cluster selection [17] assigns an instruction to a cluster where most of the predecessors of the instruction reside. However, the MWP heuristics do not take into account the future communication cost of a binding. The UAS algorithm as proposed by Ozer *et al*. [17] 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*. A detailed description and analysis of the

DOI: 10.1002/spe

```
x: MV .D2X R1242,R2854*

1.1MVK .S1 8,V109
1.2||MPY .M2 V70,V90,V1231
1.3||MPY .M1 V2239,V1260,V1243
1.4||MVK .S2 9,V147
1.5||ADD .L2X V1260,V1195,V65
1.6||ADD .L1X V106,V70,V102

2.1||MV .D1X V1254,V2843
2.2||MV .D2X V102,V2871

3.1MVK .S2 12,V171
3.2||MPY .M2 V1254,V147,V1255
3.3||SUB .L1X V65,V67,V108
3.4||ADD .D2X V102,V103,V107**
3.5||ADD .S1 V1230,V2843,V89
3.6||ADD .L2 V2854,V65,V95

4.1MPY .M2X V1231,V109,V1273
4.2||ADD .D2 V107,V171,V114
4.3|| ADD .D1 V109,V108,V115
4.4|| SUB .L1 V109,V108,V116
4.5|| ADD .S1X V89,V1231,V100
4.6|| SUB .S2 V107,V171,V119
4.7|| SUB .L2 V103,V2871,V118

*Scheduled in an earlier cycle
**Second operand of D unit is
specified first
```

Figure 5. UAS-MWP schedule.

schedules generated by different algorithms is available in the associated technical report by Nagpal and Srikant [15].

5. A GRAPH-MATCHING-BASED INTEGRATED SCHEDULING FRAMEWORK

Our integrated algorithm presented in the previous section is still not free from the curse of the fixed-ordering problem and may make a decision that may miss the opportunity to accommodate an instruction in a cycle that may otherwise be possible by exercising different scheduling alternatives for instructions. In this section, we describe our generic graph-matching-based framework for scheduling clustered architectures that resolves the phase-ordering and fixed-ordering problems associated with

DOI: 10.1002/spe

```
1.1MVK .S1 8,V109
1.2||MPY .M2 V70,V90,V1231
1.3||MPY .M1 V2239,V1260,V1243
1.4||MVK .S2 9, V147
1.5||ADD .L1X V1260,V1195,V65
1.6||ADD .D2X V106, V70,V102

2.1MVK .S2 12,V171
2.2||MPY .M2 V1254,V147,V1255
2.3||SUB .L1 V65,V67,V108
2.4||ADD .D2 V102,V103,V107
2.5||ADD .S1X V1230,V1254,V89
2.6||ADD .D1 V1242,V65,V95

3.1|| MV .L2X V89,V149
3.2|| MV .L1X V103,V150

4.1MPY .M2X V1231,V109,V1273
4.2||ADD .D2 V107,V171,V114
4.3|| ADD.D1 V109,V108,V115
4.4|| SUB .L1 V109,V108,V116
4.5|| ADD .L2 V149,V1231,V100
4.6|| SUB .S2 V107,V171,V119
4.7|| SUB.S1X V150,V102,V118
```

Figure 6. Pre-partitioning schedule.

scheduling clustered VLIW architectures [15]. A graph-matching-based scheduler further improves over the integrated algorithm by considering all the possible scheduling alternatives obtained by varying 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 the current cycle while exploiting the communication facility and parallelism offered by the 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 the maximum number of instructions in each cycle. The framework provides a mechanism to exploit the slack of instructions by dynamically varying the freedom available in scheduling an instruction and hence the cost of scheduling an instruction using different alternatives to reduce the ICC without affecting the overall schedule length. It considers the possible mapping of instructions to resources

[‡]We define the *uncovering factor* of an instruction as the number of instructions dependent on this instruction.

DOI: 10.1002/spe

in different clusters and can easily accommodates architecture-specific constraints typically found in clustered architectures. Our graph-matching-based scheduler attains approximately 28.5% and 16% performance improvement over an earlier phase-decoupled [16] and phase-coupled [17] approach to scheduling, respectively. The proposed framework is generic in terms of the number of clusters and numbers and types of functional units in each cluster, and requires only a slight tweaking of heuristics to obtain optimal performance for different clustered VLIW configurations and ICC models.

5.1. The Algorithm

Our graph-matching-based scheduler considers the instructions that 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. This is because there can be more than one alternative by which we can transfer the operands from other cluster to the target cluster (snooping, explicit MV, or combination of both) and these alternatives will vary in cost. Each edge is assigned a cost determined by a cost function and the information regarding the 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 formulate and solve a minimum cost feasible matching problem for this graph using an integer linear programming (IntLP) solver [25]. The solution gives the minimum cost feasible scheduling alternatives for the given graph. It should be noted that 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. The communication cost factor also becomes accurate from one cycle to another (as more and more parents of successors are scheduled). So, the cost of a real edge can be equal to the sum of two or more edges. Thus, a second problem is set up such that the space of all feasible matchings with the same cost as that of the minimum cost previously obtained is searched and its solution maximizes the number of instructions in the match. Instructions are considered for scheduling using the alternative dictated by the selected match. To further reduce the communication and associated overheads, 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. Algorithm 1 gives a very high-level description of our algorithm. In the following sections we describe each of these steps in detail.

5.1.1. Measuring instruction freedom

The 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 freedom of instructions are calculated assuming an infinite resource machine and thus ignoring all the resource constraints because the exact measurement 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 freedom because delay resulting from limited resources is not taken into account. A better estimate of instruction freedom is important for efficient

DOI: 10.1002/spe

Algorithm 1 Graph-matching-based spatial and temporal scheduling 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.solveMinimumCostIntLP()
  S=M.solveMaxNodeIntLP(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
```

binding and exploring the trade-off in code size and performance. For better estimation, we first schedule on an 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 on the base VLIW processor is the same as that given in Figure 8 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 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.

5.1.2. The graph construction

Algorithm 2 gives an outline of 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 in Section 5.1.4. A more detailed description of graph construction is available in the associated technical report by Nagpal and Srikant [15].

DOI: 10.1002/spe

Algorithm 2 Outline of the graph construction algorithm

```
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 i on resource r 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
```

5.1.3. IntLP formulation of the 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. Although the traditional graph-matching problem has a polynomial solution, the additional cross-path constraints that require an integer solution may make this graph matching problem an NP-complete problem (a formal proof of this is part of our future work). We have formulated the problem as a sequence of two IntLP problems. The first IntLP problem finds a feasible minimum cost match. The second IntLP problem maximizes the number of non-dummy nodes matched for the minimum cost obtained by solving the earlier problem. Formulating the problem as an ILP and solving it using an ILP solver also provides the freedom of adding other complex architecture specific constraint, if required, while porting the scheduler to some new architecture. A formal specification is depicted in Figure 7 (refer to Table I for notation).

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.

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

1. maximize the number of real edges in the final match;

DOI: 10.1002/spe

Table I. Notation used in ILP formulation.

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 pth cross-path while scheduling i on r with alternative a
N_p	Bandwidth of pth 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

minimize

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

subject to:

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

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

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

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:

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

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

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

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

Figure 7. IntLP formulation of the problem.

2. total cost of final match is not more than minimum cost of matching obtained by solving the first IntLP;
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.

DOI: 10.1002/spe

5.1.4. *The cost function*

The cost function computes the cost of scheduling an instruction on a resource using a given communication alternative. Some important parameters governing the cost are mobility of the instruction, the number and type of communication required, and the uncovering factor of the instruction. As we mentioned earlier, a better estimate of the mobility of instructions is obtained by first scheduling on a base VLIW configuration since the 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 account 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 the scheduling each instruction in the next cycle. The communication cost models proposed here is generic enough to handle different ICC models. The communication vector is composed of triple (*explicit_mv,current_comm,future_comm*), the components of which are same as those described in Section 4.1.2.

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 smaller cost are given priority over alternatives with higher cost. A communication factor is used to resolve the resource conflict among alternatives having the same mobility, while mobility is used to decide among alternatives having the same communication cost. The uncovering factor is given relatively little contribution and is used for breaking ties in the cost function. 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 instructions which have consumed 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 are scheduled, the communication cost factor become more accurate and it becomes more likely that the instruction will be scheduled in a cluster that reduces the need for future communication. Figure 8 presents one set of values that we have used during our experiments. The given values are for an architecture that has a limited snooping facility available and thus *current_comm* is given half the cost of *explicit_mv*. *future_comm* is also assigned a smaller cost, optimistically assuming that most of them will be accommodated in the free-of-cost communication slot.

5.1.5. *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 ICC and extra code added, the scheduling of instructions having high communication cost and enough freedom can be deferred to future cycles in the hope of scheduling them later using

DOI: 10.1002/spe

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
$\alpha = 1.00$, $\beta = 1.00$, X = 0.5, Y = 0.75, Z = 1.00, A = 1.00, B = 1.00, C = 0.10

Figure 8. Cost function.

1. MVK 8,V109
2. MVK 9,V147
3. MPY V70,V90,V1231
4. MPY V2239,V1260,V1243
5. MPY V70,V60,V243
6. ADD V1260,V1195,V65
7. ADD V106,V70,V102
8. ADD V1512,V60,V115
9. SUB V124,V60,V112
10. SUB V124,V2239,V118
11. SUB V1512,V80,V181

Figure 9. Ready instructions.

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 the next section.

5.2. An example

Let us assume that the earlier scheduling decisions were such that the V2239, V1260, V106, V1512, and V124 are bound to cluster 1 while V70, V90, V1195, V60, and V80 are bound to cluster 2. Figure 9 lists the set of instructions ready to be scheduled in the current cycle. Tables II and III enumerate the scheduling alternatives possible for each instruction along with the number of explicit MVs and snoops used by an alternative and future communication that will arise by exercising the alternative. Scheduling alternatives for an instruction vary in terms of functional unit and communication options used (explicit move, snooping facility, or a combination of both). The commutativity of operations such as ADD and MPY is taken into account while considering the possible scheduling alternatives.

DOI: 10.1002/spe

Table II. Scheduling alternatives for instructions: L1, S1, D1, and M1.

Number	Instructions	Mobility	Number of consumers	L1			S1			D1			M1		
				E	S	F	E	S	F	E	S	F	E	S	F
1.1	MVK	4	1				0	0	0						
2.1	MVK	4	1				0	0	1						
3.1	MPY	2	2										1	1	0
3.2													2	0	0
4.1	MPY	2	2										0	0	0
4.2															
5.1	MPY	2	1												
5.2															
6.1	ADD	3	2	0	1	0	0	1	0	0	1	0			
6.2				1	0	0	1	0	0	1	0	0			
7.1	ADD	3	2	0	1	1	0	1	1	0	1	1			
7.2				1	0	1	1	0	1	1	0	1			
8.1	ADD	5	1	0	1	0	0	1	0	0	1	0			
9.1	SUB	4	2	1	0	1	1	0	1						
10.1	SUB	4	2	0	0	0	0	0	0						
10.2															
11.1	SUB	4	1	1	0	1	1	0	1						
11.2				0	1	1	0	1	1						

Legend: E, explicit MVs; S, snoops; F, future communications; MF, mobility factor; UF, uncovering factor; CF, communication cost factor; TC, total cost.

We enumerate all the possible scheduling alternatives for all instructions and leave it to the matcher to select among the scheduling alternatives with an aim of scheduling as many instructions as possible in each cycle. Tables IV and V present the mobility factor, uncovering factor, communication cost factor, and total cost computed for all scheduling alternatives using the cost function described above. Figure 10 depicts the partial matching graph having an edge for each possible scheduling alternative of the given set of instructions (dummy edges are not shown to preserve the clarity).

Figure 11 presents the schedule generated by the graph-matching-based scheduler and Figure 12 presents a possible schedule generated using the UAS algorithm with MWP heuristic for cluster selection. Each instruction in the generated schedule is suffixed with the scheduling alternative number used by the scheduler. A detailed description and analysis of the scheduled generated by the graph-matching-based scheduler and UAS is available in the associated technical report by Nagpal and Srikant [15].

6. EXPERIMENTAL RESULTS

6.1. Experimental setup

We have used the SUIF compiler [18] along with MACHSUIF library [19] for our experimentation. The C source file is fed to the SUIF front end that generates the SUIF IR. We perform a number

DOI: 10.1002/spe

Table III. Scheduling alternatives for instructions: L2, S2, D2, and M2 (see Table II for legend).

				L2			S2			D2			M2		
Number	Instruction	Mobility	Number of consumers	E	S	F	E	S	F	E	S	F	E	S	F
1.1	MVK	4	1							0	0	0			
2.1	MVK	4	1							0	0	0			
3.1	MPY	2	2										0	0	0
3.2															
4.1	MPY	2	2										1	1	0
4.2													2	0	0
5.1	MPY	2	1												
5.2															
6.1	ADD	3	2	0	1	1	0	1	1	0	1	1			
6.2				1	0	1	1	0	1	1	0	1			
7.1	ADD	3	2	0	1	0	0	1	0	0	1	0			
7.2				1	0	0	1	0	0	1	0	0			
8.1	ADD	5	1	0	1	1	0	1	1	0	1	1			
9.1	SUB	4	2	1	0	0	1	0	0						
10.1	SUB	4	2	1	1	1	1	1	1						
10.2				2	0	1	2	0	1						
11.1	SUB	4	1	1	0	0	1	0	0						
11.2				0	1	0									

of classical optimizations[§] to get a highly optimized intermediate representation. This is followed by lowering of code level. Optimizations are again applied on the lower level code as lowering produces additional opportunities for optimization. We perform most of the optimizations on both high-level and low-level intermediate code. We generate code for TMS320C6X [26]. Ours is a hand-crafted code generator that is designed to take advantage of the varied 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. Our scheduler is parameterizable by a machine description file which specifies following attributes of the target machine [27,28]:

- the number of clusters, cross-paths among clusters, and available load/store paths for each cluster;
- the number and types of resources in each cluster as well as latency of resources;
- the number and types of registers in each cluster;
- the operations formats, scheduling alternatives, and resource usage of each operation.

[§]List of optimizations includes constant folding, constant propagation, copy propagation, common subexpression elimination, dead-code elimination, loop invariant code motion, induction variable elimination, control flow simplification, if hoisting, and many more.

DOI: 10.1002/spe

Table IV. Cost of scheduling alternatives: L1, S1, D1, and M1 (see Table II for legend).

				L1		S1		D1		M1	
Number	Instruction	MF	UF	CF	TC	CF	TC	CF	TC	CF	TC
1.1	MVK	0.67	0.03			0	0.7				
2.1	MVK	0.67	0.03			0.27	0.97				
3.1	MPY	0.33	0.0							0.55	0.88
3.2										0.73	1.06
4.1	MPY	0.33	0.0							0.0	0.33
4.2											
5.1	MPY	0.33	0.03							0.55	0.91
5.2										0.73	1.09
6.1	ADD	0.5	0.0	0.18	0.68	0.18	0.68	0.18	0.68		
6.2				0.36	0.86	0.36	0.86	0.36	0.86		
7.1	ADD	0.5	0.0	0.45	0.95	0.45	0.95	0.45	0.95		
7.2				0.64	1.14	0.64	1.14	0.64	1.14		
8.1	ADD	0.83	0.03	0.18	1.05	0.18	1.05	0.18	1.05		
9.1	SUB	0.67	0.0	0.64	1.3	0.64	1.3				
10.1	SUB	0.67	0.0	0.0	0.67	0.0	0.67				
10.2											
11.1	SUB	0.67	0.03	0.64	1.34	0.64	1.34				
11.2				0.45	1.15	0.45	1.15				

Table V. Cost of scheduling alternatives: L2, S2, D2, and M2 (see Table II for legend).

				L2		S2		D2		M2	
Number	Instruction	MF	UF	CF	TC	CF	TC	CF	TC	CF	TC
1.1	MVK	0.67	0.03			0	0.7				
2.1	MVK	0.67	0.03			0	0.7				
3.1	MPY	0.33	0.0							0	0.33
3.2											
4.1	MPY	0.33	0.0							0.55	0.88
4.2										0.73	1.06
5.1	MPY	0.33	0.03							0	0.37
5.2											
6.1	ADD	0.5	0.0	0.45	0.95	0.45	0.95	0.45	0.95		
6.2				0.64	1.14	0.64	1.14	0.64	1.14		
7.1	ADD	0.5	0.0	0.18	0.68	0.18	0.68	0.18	0.68		
7.2				0.36	0.86	0.36	0.86	0.36	0.86		
8.1	ADD	0.83	0.03	0.45	1.32	0.45	1.32	0.45	1.32		
9.1	SUB	0.67	0.0	0.36	1.03	0.36	1.03				
10.1	SUB	0.67	0.0	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	0.36	1.06	0.36	1.06				
11.2				0.18	0.88						

DOI: 10.1002/spe

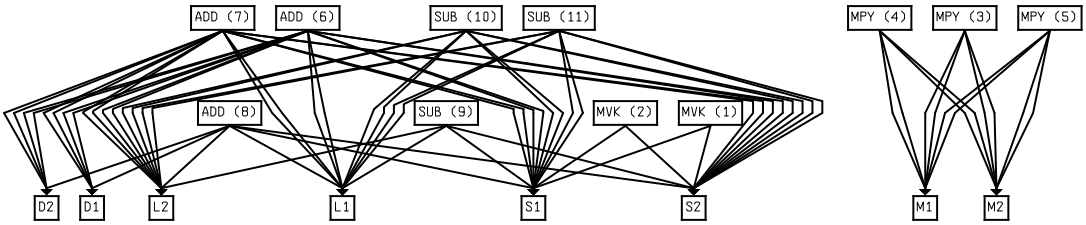

Figure 10. Partial matching graph.

```
MV .D2X V106, V2244*

C1.3||MPY .M2 V70,V90,V1231 (3.1)
C1.4||MPY .M1 V2239,V1260,V1243 (4.1)
C1.5||ADD .D1X V1260,V1195,V65 (6.1)
C1.6||ADD .D2X V2244,V70,V102 (7.2)
C1.7||SUB .L2X V124,V60,V112 (9.1)
C1.8||SUB .L1 V124,V2239,V118 (10.1)
C1.1||MVK .S1 8, V109 (1.1)
C1.2||MVK .S2 9, V147 (2.1)

* Scheduled in an earlier cycle
```

Figure 11. Schedule generated using graph matching without selective rejection (GM).

We have interfaced our scheduler with the CPLEX IntLP solver [25] (using the good set of API functions provided by the CPLEX suite) to handle the IntLP formulation of the graph matching problem. The input to the IntLP solver consists of various scheduling alternatives of instructions and the IntLP solver returns the best match as dictated by our matching criteria. The encoding of input and output to the CPLEX solver [25] is done using a file format designed by us. 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 [29]. 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 a simulation of the generated code using the TI simulator for the TMS320C6X architecture [26].

DOI: 10.1002/spe

```
C1.1||MV .D1X V80,V2246
C1.2||MV .D2X V124,V2325

C2.1||MPY .M2 V70,V90,V1231 (3.1)
C2.2||MPY .M1 V2239,V1260,V1243 (4.1)
C2.3||ADD .L1X V1260,V1195,V65 (6.1)
C2.4||ADD .L2X V106,V70,V102 (7.1)
C2.5||SUB .S2X V2325,V60,V112*
C2.6||SUB .S1 V124,V2239,V118 (10.1)
C2.7||ADD .D1X V1512,VV2246,V181*

* Scheduled using and explicit MV in a separate cycle
```

Figure 12. Schedule generated using unified assign and schedule with MWP (UM).

6.1.1. Benchmark programs

Table VI summarizes the key characteristics of benchmark programs that we have used for the experimental evaluation of our framework and a comparison with the pre-partitioning algorithm [16] and UAS [17]. These benchmarks are mostly unrolled inner loop kernels of MEDIABENCH [30], a representative benchmark of multimedia and communication applications and is specially designed for embedded applications. It is important to note that these inner loop kernels are reasonably large after unrolling and contain thousands of instructions. Any further ILP enhancement technique such as superblock or hyperblock formation will further help our scheduler to generate better schedules. We have selected kernels from all the major embedded application areas. These include filter programs such as complex FIR, Cascaded IIR, and autocorrelation, imaging routines such as wavelet transformation, quantization, and shading and some transformation routines such as FFT and DCT. Some programs from the telecommunication domain such as viterbi decoder, convolution decoder, etc., have also been used. Table VI 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. The L/N ratio is an approximate measure of the available ILP in the program. When the L/N ratio 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 0.09 and 0.46.

6.2. Performance results

6.2.1. Execution time and code size

Figure 13 depicts the speed-up of different algorithms as compared with the integrated scheduling algorithm. Figure 14 presents the percentage distribution of explicit inter-cluster MV instructions

DOI: 10.1002/spe

Table VI. Benchmark programs.

Name	Description	L/N
AUTOCORR	Auto correlation	0.15
CORR 3x3	3x3 correlation with rounding	0.11
FIR	Finite impulse response filter	0.12
FIRCX	Complex FIR	0.09
IIRCAS	Cascaded biquad IIR filter	0.18
GOURAUD	Gouraud shading	0.46
MAD	Minimum absolute difference	0.17
QUANTIZE	Matrix quantization	0.38
WAVELET	1D wavelet transform	0.16
IDCT	IEEE 1180 compliant IDCT	0.09
FFT	Fast fourier transform	0.19
BITREV	Bit reversal	0.31
VITERBI	Viterbi decoder	0.17
CONVDOC	Convolution decoder	0.20

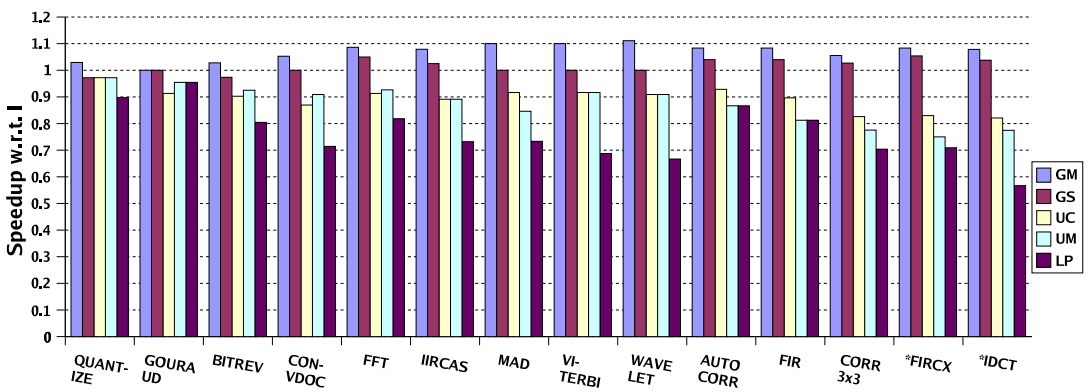

Figure 13. Speedup as compared with the integrated scheduling algorithm (I): GM, graph matching without selective rejection; GS, graph matching with selective rejection; UAS, unified assign and schedule [17]; MWP, magnitude weighted predecessors ordering [17]; CWP, completion weighted predecessors ordering [17]; UC, unified assign and schedule with CWP [17]; UM, unified assign and schedule with MWP [17]; LP, Lapinskii's pre-partitioning algorithm [16].

DOI: 10.1002/spe

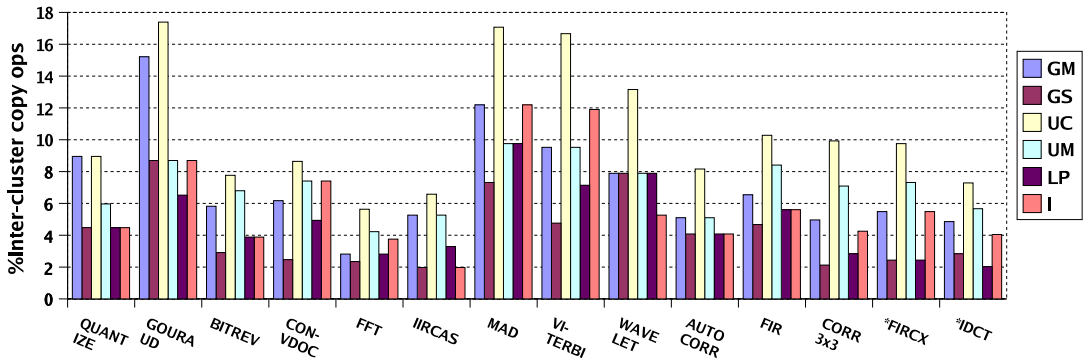

Figure 14. Percentage distribution of explicit inter-cluster MV instructions versus other instructions (see Figure 13 for legend).

versus other instructions for different algorithms. We have experimented with two main heuristics for considering clusters namely CWP and MWP as proposed by Ozer *et al*. [17]. 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 with GM. The integrated algorithm could attain approximately 10.67%, 12.64%, and 23.80% improvement, respectively, over UC, UM and LP. On the average, the percentage of explicit inter-cluster MVs versus other instructions is 4.21%, 4.84%, 5.93%, 7.20%, 7.08%, and 10.52% for GS, LP, I, GM, UM, and UC, respectively. Tables VII and VIII summarize all the results.

The graph-matching-based scheduler performs best in terms of execution time. However, it introduces more explicit MV operations. This is because it is geared towards scheduling an instruction in the current cycle even if it possesses enough freedom and requires explicit MV operation. GS reduces the number of explicit MV operations by incorporating a selective rejection mechanism with some performance degradation attributed to operation serialization in the later cycles. Our integrated algorithm provides performance comparable to GS and code size penalty (in terms of explicit MV operations) slightly higher than GS and LP. The integrated algorithm 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 either a medium or small 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.

GS and LP perform better than the other algorithms in terms of the number of explicit MV operations used for ICC. LP, having a global view of DFG and tending to reduce ICC, incurs less code size penalty in terms of explicit MV operations. GM is better than UM and UC, and UC performs worse of all. The integrated algorithm performs consistently better than UC and UM and slightly worse than GS and LP in terms of the number of explicit MV instructions used for ICC.

DOI: 10.1002/spe

Table VII. Speedup comparison.

	GS	I	UC	UM	LP
GM	4.97%	6.4%	16.37%	18.16%	28.51%
GS		1.48%	11.94%	13.82%	24.87%
I			10.67%	12.64%	23.80%
UC				2.29%	14.87%
UM					12.73%

Table VIII. Average %Inter-cluster MV.

GS	LP	I	GM	UM	UC
4.21%	4.84%	5.93%	7.20%	7.08 %	10.52%

6.2.2. Compilation time

Although we formulate and solve the problem as a sequence of two IntLP problems, the compilation time is still of the order of milliseconds even for reasonable large DFG. We have solved the problem using hundreds of nodes (for a eight-wide machine) in a cycle and were able to schedule the whole DFG in the order of milliseconds. This is because the problem is solved for set of instructions ready to be scheduled in a particular cycle. Thus even considering a practical 16-wide machine (with a 4 or 8 cluster configuration) having an additional communication constraint will not increase the compilation time to an extent that may question the practicability of the approach given the quality of generated schedules and the importance of quality code in the embedded domain.

6.3. Performance evaluation

In this section we discuss the performance of GM, GS, I, and our implementation of the UAS algorithm (with CWP and MWP ordering for cluster selection) [17] and Lapinskii's pre-partitioning algorithm [16].

As mentioned earlier (see Section 2.5), we consider a fully pipelined machine model where each resource can initiate an operation every cycle. Resources vary in terms of their operational and communicative capabilities and the snooping 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 lack of) knowledge of the above facts and tending to reduce ICC make spatial decisions which artificially constrain the temporal scheduler in the later phase. These algorithms thus suffer from 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.

DOI: 10.1002/spe

Although these algorithms may able to reduce the ICC by working on a global view of the DFG, we observe that it is often done at the cost of performance.

UAS scheme proposed by Ozer *et al*. [17] integrates cluster assignment into the list scheduling algorithm and shows improved performance over phase decoupled scheduling approaches. They have proposed many orders for considering clusters. However, they do 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 generated on clustered architectures in general and the machine model under consideration in particular. 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 owing to a binding and this may lead to a stretched schedule resulting from the generation of more communications than the available bandwidth can accommodate in a cycle and hence a greater number of explicit move operations. An explicit move operation reserves a resource for a cycle on the machine model under consideration and this may lead to poor resource utilization separate from increases in register pressure and code size. These are the reasons why UAS suffers from performance and code size penalties. 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.

Our integrated scheduling algorithm shows improvement over UAS by effectively exploiting the information regarding free cross-paths and functional units available in the current cycle as well as in the earlier cycles and also the number and types of operations ready to be scheduled in the current cycle and their resource requirements. Utilizing this precise knowledge available at schedule time, our algorithm tries to schedule an operation in the current cycle while minimizing the communication in the current as well as future cycles. *Our integrated scheme strives to bind an operation to a functional unit in a cluster where its operand can be snooped from the other clusters rather than explicitly using the communication path.* This urge for avoiding explicit cross-cluster MV operations while maximizing utilization of the snooping capability helps to reduce the extra MV instructions and consequently the reservation of functional units for MV operation, code size as well as register pressure. Functional unit binding is done while taking care of other operations to be scheduled in the current cycle and their needs for functional units and the cross-path. This helps in combating the situation where earlier naive decisions lead to unnecessary delay in scheduling an operation in the current cycle and the consequent stretch of schedule.

Our scheme is very aggressive in utilizing the free cross-path and tries to schedule an instruction in the current cycle by trying to schedule communication in the earlier cycles using the free communication slot wherever necessary. Thus in some cases the integrated scheme introduces more explicit MV operations. However, the extra code size (due to explicit MV operations) in most of the cases is the same as that of the pre-partitioning scheme and is only marginally more for some benchmarks. The primary reason for this is that the algorithm prefers to snoop operands from other clusters rather than simply inserting a MV operation.

Although our integrated algorithm offers an improvement over UAS by taking into account future communication that may arise owing to a binding and a better functional unit binding mechanism, it still suffers from the fixed-ordering problem and may make a decision that may miss the opportunity to accommodate an instruction in a cycle which may otherwise be possible by exercising different

DOI: 10.1002/spe

scheduling alternatives for instructions. GM further improves over the integrated algorithm by considering all the possible scheduling alternatives obtained by varying 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 the current cycle while exploiting the communication facility and parallelism offered by the hardware. A cost function composed of various dynamically varying factors such as the communication cost, the freedom available in scheduling the instruction, and the uncovering factor of the instruction is used to select from scheduling alternatives for instructions competing for limited resources while scheduling the maximum number of instructions in each cycle. Instructions with smaller freedom are scheduled in preference to those with higher freedom values despite their high communication cost, in order to avoid stretching the overall schedule. Scheduling alternatives for instructions with enough freedom and high communication cost are assigned a high cost so as to favor 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.

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 ICC and associated overheads, GS incorporates a 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. Thus it provides a mechanism to explore the trade-off in code size and performance. However, performance results shows that selective rejection incurs performance penalty to reduce the code size.

7. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we considered a generic and realistic clustered machine model that is based on commercial clustered architectures. We proposed an integrated scheduling algorithm targeting such a machine model and experimentally demonstrated its superiority over a few other proposed algorithms. We also proposed a generic graph-matching-based framework that resolves the phase-ordering and fixed-ordering problems associated with scheduling a clustered VLIW datapath and experimentally demonstrated its properties. Our scheduling framework can be extended in many different ways.

1. Although we concentrated only on acyclic scheduling, in future work the proposed framework could be adapted for software pipelining of inner loops and compared with other existing algorithms for software pipelining.
2. We observe that the number of instructions per cycle (IPC) varies from one cycle to another. This variation in IPC can be exploited together with the slack of instructions to develop a power-sensitive scheduling framework for exploring trade-offs in leakage-power consumption and execution time.

REFERENCES

1. Ramakrishna Rau B, Fisher JA. Instruction-level parallel processing: History, overview, and perspective. *Journal of Supercomputing* 1993; **7**(1–2):9–50.

DOI: 10.1002/spe

2. Johnson W. *Superscalar Microprocessor Design*. Prentice-Hall: Englewood Cliffs, NJ, 1991.
3. Fisher JA. Very long instruction word architectures and the ELI-512. *25 years of the International Symposia on Computer Architecture (selected papers)*. ACM Press: New York, 1998; 263–273.
4. Palacharla S, Jouppi NP, Smith JE. Complexity-effective superscalar processors. *Proceedings of the 24th International Symposium on Computer Architecture*. ACM Press: New York, 1997; 206–218.
5. Matzke D. Will physical scalability sabotage performance gains. *IEEE Computer* 1997; **30**(9):37–39.
6. Ho R, Mai K, Horowitz M. The future of wires. *Proceedings of the IEEE* 2001; **89**(4):490–504.
7. Capitanio A, Dutt N, Nicolau A. Partitioned register files for VLIWs: A preliminary analysis of tradeoffs. *Proceedings of the 25th International Symposium on Microarchitecture*. IEEE Computer Society Press: Los Alamitos, CA, 1992; 292–300.
8. Farkas KI, Chow P, Jouppi NP, Vranesic Z. The multicluster architecture: Reducing cycle time through partitioning. *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society: Los Alamitos, CA, 1997; 149–159.
9. Faraboschi P, Brown G, Fisher JA, Desoli G. Clustered instruction-level parallel processors. *Technical Report*, Hewlett-Packard, 1998.
10. Terechko A, Le Thenaff E, Garg M, van Eijndhoven J, Corporaal H. Inter-cluster communication models for clustered VLIW processors. *Proceedings of Symposium on High Performance Computer Architectures*, February 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003.
11. Texas Instruments Inc. Tms320c6000: A high performance dsp platform. http://www.ti.com/sc/docs/products/dsp/ [August 2003].
12. Faraboschi P, Brown G, Fisher JA, Desoli G, Homewood F. Lx: A technology platform for customizable VLIW embedded processing. *Proceedings of the 27th annual International Symposium on Computer Architecture*. ACM Press: New York, 2000; 203–213.
13. Fridman J, Greefield Z. The tigersharc DSP architecture. *IEEE Micro* 2000; **20**(1):66–76.
14. Pechanek GG, Vassiliadis S. The manarray embedded processor architecture. *Proceedings of the 26th Euromicro Conference*. IEEE Computer Society Press: Los Alamitos, CA, 2000; 348–355.
15. Nagpal R, Srikant YN. Pragmatic integrated scheduling for clustered VLIW architectures. *Technical Report*, Department of CSA, Indian Institute of Science, 2007. Available at: http://www.archive.csa.iisc.ernet.in/TR [August 2003].
16. Lapinskii VS, Jacome MF, De Veciana GA. Cluster assignment for high-performance embedded VLIW processors. *ACM Transactions on Design Automation of Electronic Systems* 2002; **7**(3):430–454.
17. Ozer E, Banerjia S, Conte TM. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures. *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society Press: Los Alamitos, CA, 1998; 308–315.
18. SUIF Compiler System. http://suif.stanford.edu/ [March 2007].
19. MACHINE SUIF. http://www.eecs.harvard.edu/hube/software/software.html [August 2003].
20. Lee W, Puppin D, Swenson S, Amarasinghe S. Convergent scheduling. *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society Press: Los Alamitos, CA, 2002; 111–122.
21. Seshan N. High VelociTI processing. *IEEE Signal Processing Magazine* 1998; **15**(2).
22. Leupers R. Instruction scheduling for clustered VLIW DSPs. *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Philadelphia, PA, October 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
23. Kailas K, Agrawala A, Ebcioglu K. CARS: A new code generation framework for clustered ILP processors. *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA'01)*, January 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
24. Zalamea J, Llosa J, Ayguade E, Valero M. Modulo scheduling with integrated register spilling for clustered VLIW architectures. *Proceedings of the 34th Annual International Symposium on Microarchitecture*, December 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001; 160–169.
25. CPLEX, Using the CPLEX callable library Ver3, 1995. http://www.ilog.com/products/cplex/ [August 2003].
26. Texas Instruments Inc. TMS320C6000 CPU and Instruction Set reference Guide, 1998. http://www.ti.com/sc/docs/products/dsp/ [August 2003].
27. Srikant YN. Shankar P (eds.). *The Compiler Design Handbook: Optimizations and Machine Code Generation*. CRC Press: Boca Raton, FL, 2002.
28. Qin W, Malik S. Architecture description languages for retargetable compilation. *The Compiler Design Handbook: Optimizations and Machine Code Generation*. CRC Press: Boca Raton, FL, 2002; 535–564.
29. Chow FC, Hennessy JL. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems* 1990; **12**(4):501–536.
30. Lee C, Potkonjak M, Mangione-Smith WH. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society: Los Alamitos, CA, 1997; 330–335.

DOI: 10.1002/spe