Hierarchical Cluster Assignment for Coarse-Grain Reconfigurable Coprocessors

Martino Sykora[1], Davide Pavoni, Joel Cambonie[2], Roberto Costa[3] and Stefano Crespi Reghizzi[1]

[1]Politecnico di Milano
Dip. Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano, Italy
{sykora,crespi}@elet.polimi.it

[2]STMicroelectronics - Grenoble
12, rue J. Horowitz - B.P. 217
F-38019 Grenoble, France
joel.cambonie@st.com

[3]STMicroelectronics - AST Manno Lab
Via Cantonale 16/E
6928, Manno, Switzerland
roberto.costa@st.com

Abstract

Embedded media applications have to satisfy real-time, low power consumption and silicon area constraints. These applications spend most of the execution time in the iteration of a few kernels; such kernels are typically made of independent operations, which can be executed in parallel. Clustered architectures are a solution designed to exploit the high Instruction Level Parallelism (ILP) of the media kernels, to keep a good level of scalability and to match the strict constraints of the embedded domains. Within this category, architectures with reconfigurable connections between clusters are of particular interest. The enhanced flexibility allows them to handle several different data-paths effectively, hence multiple applications; this is a key economic factor in the semiconductor world, in which the cost of the masks significantly increases at every technological advance. This papers describes Hierarchical Cluster Assignment (HCA), a compilation technique that deals with the problem of mapping the computation of multimedia kernels onto the clusters of the target machine. HCA exploits the hierarchical structure of the clusters of the target architectures; it works by decomposing the problem of cluster assignment into a sequence of simpler sub-problems, each of them involving a subset of the kernel instructions and a subset of the machine clusters. A prototype of this methodology has been implemented in a flexible framework and tested on machine models based on the DSPFabric architecture.

1 Introduction

Embedded multimedia applications are subject to strong real-time and QoS constraints. Their implementation on portable devices is challenging, since it simultaneously needs to achieve high performance, low power consumption, low cost and fast time-to-market. In order to avoid the non-recurring costs of ASIC design, industry and academia invested resources in the research for coarse grain reconfigurable architectures throughout the last fifteen years.

The hardware maturity, the increasing demand of computational power by multimedia applications and the aggressive competition in the market of media streaming are established facts during the last few years, which make coarse-grain reconfigurable hardware an attractive solution for the execution of application kernels.

1-4244-0910-1/07/$20.00 ©2007 IEEE.

Media applications, like audio and video encoding/decoding, often spend most of the execution time in the iteration of a few kernels, i.e. Inverse Discrete Cosine Transform (IDCT), interpolation or deblocking filters. These kernels are largely made of independent operations and low memory aliasing, thus exposing a high degree of Instruction Level Parallelism (ILP).

Clustered architectures – where each cluster is a Processing Element (PE) with its own functional units and register bank – allow exploiting the ILP and guarantee scalability, low power consumption and low cost. Equivalent unified machines (same amount of resources in a single cluster) are not feasible because of the more than linear increase of the centralized register file read/write time, w.r.t. the increase of the functional units accessing it. On one hand, clustered architectures are a solution to achieve highly parallel hardware. On the other, they suffer of non-unary delays caused by explicit inter-cluster copies. Such architectures are said *homogeneous* if all their clusters present the same kind of functional units, *heterogeneous* otherwise. The migration of the operands among clusters is demanded to a word-level communication network and is controlled by special instructions – like `snd` or `rcv` – executed by the clusters themselves, or by dedicated hardware.

These architectures belong to the family of Scalar Operand Networks (SON), and can be characterized by the AsTrO taxonomy [23], which specifies whether the assignment of the instructions, the transport of the operands and the ordering of the instructions are statically or dynamically performed.

Coarse-grain reconfigurability adds another dimension to the design space and provides the architecture with high flexibility. It makes possible to instantiate a specific data-path among the clusters, by selecting a subset of available connections, compatible with the communication constraints.

The key compilation technology to exploit this class of architectures at their best is the *Instruction Cluster Assignment* (ICA). During the ICA, the compiler partitions the instructions of the Data Dependency Graph (DDG) over the clusters, trying to reach a balance between the conflicting goals of high parallelism and low penalties caused by intercluster copies. More precisely, the compiler performs the ICA pass by optimizing a global cost function, built on a set of heuristic criteria. Such criteria are aimed at achieving the best compromise between those opposite goals; to be effective, they also estimate and take into account parameters that come into play in later phases of the compilation flow. For instance, if the scheduling technique is Modulo Scheduling [20], the cost function must take into account the Initiation Interval (II), the register pressure and the lifetime of the temporaries. Moreover, the datapath correctness must be guaranteed by the compiler, which selects the communication patterns to be configured, compatibly with the limited amount of input/output ports w.r.t. to total amount of potential connections. Consequently the ICA pass is critical both to produce legal code and to obtain a good quality schedule at the end of the compilation flow.

This paper presents the original development of Hierarchical Cluster Assignment (HCA), a methodology specifically conceived for architectures having the inter-connection network with a hierarchical structure. This technique divides the ICA problem into a set of multiple interacting problems, each of them addressing a specific level of the hardware hierarchy. Each problem, if considered by itself, is an ICA pass that maps a portion of the DDG on a non-hierarchical machine. The hierarchical nature is highlighted in the structure of the constraints, which flow through sequential problems. The main appeal of this approach is that it easily scales with the architecture and presents a regular and modular structure, two fundamental qualities in a compiler design.

We consider Reconfigurable Co-Processor (RCP) [6] and DSPFabric [3] by STMicroelectronics as target architectures. The former does not present a hierarchical nature; however, it allows introducing some concepts valid also for the DSPFabric architecture (whose inter-connection network is strongly hierarchical). Both RCP and DSPFabric are designed for supporting Kernel Only Modulo Scheduled [21] loops. In this paper we focus on the HCA side only, considering the II of the loop as the main cost factor of the goal function. The present choice of the objective function does not preclude more realistic choices in the future, to take into account scheduling aware details, as register pressure and lifetimes. We have integrated HCA in a flexible prototype, designed to perform ICA and scheduling over a wide range of architectures.

To the best of our knowledge, the only work on hierarchical cluster assignment technique, specifically tailored for hierarchical inter-connection networks, was done for the implementation of the CADDI compiler. It generates code for PADDI-2 [24] machine, which shares several architectural aspects with DSPFabric. However, the authors report scarce details. Chu *et al.* [4] proposed a methodology for Hierarchical Operation Partitioning, based on a multilevel clustering pass, followed by a step of solution improvement. However, their hierarchical approach is presented as a generic ICA technique, since they do not consider neither multilevel hardware architectures nor reconfigurable topologies.

The novelty of this work is a detailed description of the algorithmic passes involved in HCA on a hierarchical and reconfigurable architecture. Moreover, we present a flexible framework prototype, conceived for the passes of cluster assignment and scheduling, which can be easily integrated in an already existing compilation back-end.

The remainder of this paper is organized as follows: Section 2 provides an overview of RCP and DSPFabric architectures. Section 3 introduces the high level organization of our cluster assignment framework. Section 4 explains the technique of HCA. Section 5 mentions the initial experiments obtained clustering significant multimedia applications over DSPFabric architecture. Section 6 reports significant related works on cluster assignment and scheduling for multiclustered and reconfigurable architectures. Conclusions and future work are discussed in the last section.

2 Architecture Overwiev

Reconfigurable Co-Processor (RCP) [6] and DSPFabric [3] are multicluster architectures specifically designed for computationally intensive loops of multimedia applications. With respect to the AsTrO taxonomy, they are Static-Static-Static (SSS) SONs [23], which means that the assignment of the instructions, the displacement of the copies and the scheduling passes are compiler tasks.

Moreover, RCP and DSPFabric are characterized by coarse-grain reconfigurable data-paths. The compiler must select a sub-set of feasible cluster connections for data flowing, and emit the reconfiguration instructions for activating the selected wires. These instructions will be executed at run time, in a reconfiguration phase that precedes the execution of the application; this induces a specific network topology.

The reconfiguration space – the space of feasible topologies – is tailored by the constraints given by the availability of I/O ports with respect to the total number of connecting wires.

2.1 RCP Architecture

In RCP architecture, each cluster could receive values from N neighbors, but the availability of only K ($K < N$) input ports per cluster limits the number of connections simultaneously configurable. Figure 1 (a) shows a 8-cluster RCP, in which each cluster could receive a copy from 4 neighbors, (b) depicts a feasible topology when the number of input ports per cluster is 2. RCP is a heterogeneous

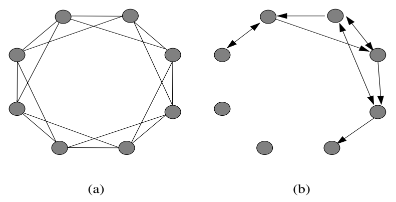


Figure 1. A 8-cluster RCP ring topology. (a) Potential connections (b) A feasible topology

machine, since only some PEs can issue instructions accessing memory. RCP shares the memory subsystem, including the caches, with the main processor.

More interesting is the DSPFabric organization, in which each cluster can be potentially connected to all the others, exploiting a hierarchical inter-connection schema, based on different levels of MUXes. Effective limitations are given by the MUXes capacity. We describe the DSPFabric architecture in the following, focusing the attention on the structure of the inter-connections.

2.2 DSPFabric Architecture

Figure 2 gives an overall picture of a 64-cluster DSPFabric Co-processor. At level 0, it can be seen as an array of four 16-issue PEs, communicating through a collection of multiplexers, which realize a multi input/output switch. Each cluster set has N input wires and N output wires, output wires being possibly broadcasted to all the others. At the contrary, input wires can be connected to only one source. Figure 3 shows a feasible data path at level 0, assuming

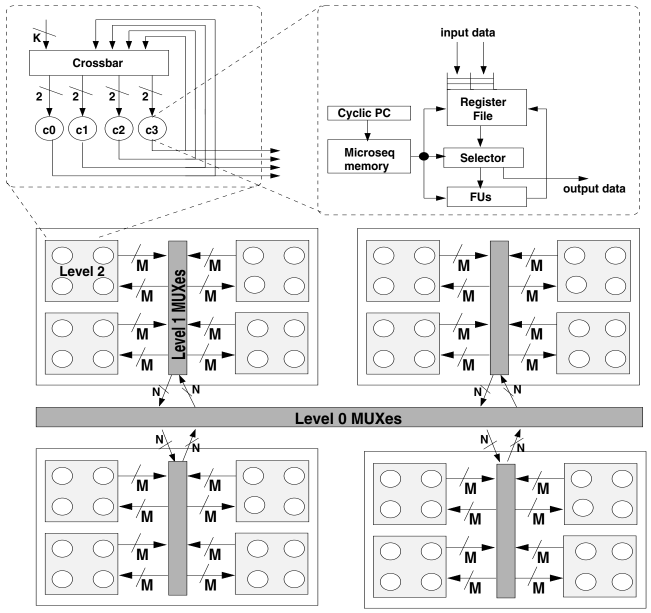


Figure 2. A 64-cluster DSPFabric architecture

N equal to 4. At level 1, the spatial structure replicates it-

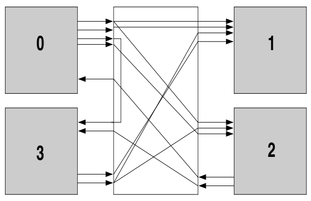


Figure 3. A feasible inter-connection among cluster sets.

self inside each set of clusters, presenting again an array of 4-issue processors, connected together by multiplexers with capacity M. The last level is composed by the *computation nodes* (CNs) connected through a reconfigurable crossbar, which takes as input the internal connections and K of the wires incoming from level 1. Each computation node has two incoming wires and one outgoing wire.

The computation nodes are single issue pipelined machines, accessing their own register file and functional units. Since DSPFabric has been specifically designed as loop accelerator Co-processor, each cluster is equipped with hardware features for better executing modulo scheduled code [20], like support for instruction predication and rotating registers. Precisely, the application is scheduled using the Kernel-Only Modulo Scheduling [21] technique, which fully predicates the loop epilogue. Thus, no branches are allowed and the execution is controlled by a cyclic program counter.

The intercluster copies are controlled by `receive` primitives executed by the destination cluster. Two regions of its register file are organized as input buffers, which push the incoming values on top, but can be read randomly by the receiver.

The coupling with the main memory subsystem is demanded to a programmable DMA. Each cluster can generate an address request, which is directly sent to DMA without consuming inter-clusters communication patterns. Only a limited number of requests can be served at the same time, e.g. 8 requests, thus the compiler must ensure that the amount of simultaneous requests does not exceed that limit. Since the memory requests have no unary latency, the DMA engine provides input and output FIFOs – of depth equal to the serving time – for handling high memory pressure. When a value is ready it is directly loaded in the requesting cluster register file.

Since DSPFabric – and, more generally, all the Co-processors designed for multimedia embedded applications – is specifically designed for performing media streaming, and the input/output streams are characterized by a highly regular structure and largely independent data, the DMA programmable interface allows to perform efficient data buffering and to mask the memory latencies.

3 Framework Organization

Figure 4 describes the organization of our cluster assignment framework, focusing on its software interfaces. We designed it to perform cluster assignment at a single level of the interconnection hierarchy, merging multiple levels as described in the next section.

The framework takes the DDG and the Pattern Graph (PG) as input; the latter represents the architecture topology at a high abstraction level. Each node of the PG – called cluster in the following – is represented by its *Resource Table* (RT); a potential physical connection between two nodes is described by an arc. At this level of abstraction, an edge just identifies a potential communication pattern, without introducing low-level details; i.e. it specifies that two clusters could be connected by a communication pattern, but it does not focus on the nature of the interconnection.

The software interfaces are shown in Figure 4 in light grey (in the following we refer to them in *italic* font). They characterize the behavior of the Space Exploration Engine (`SEE`), which aims at assigning the nodes of the DDG to the

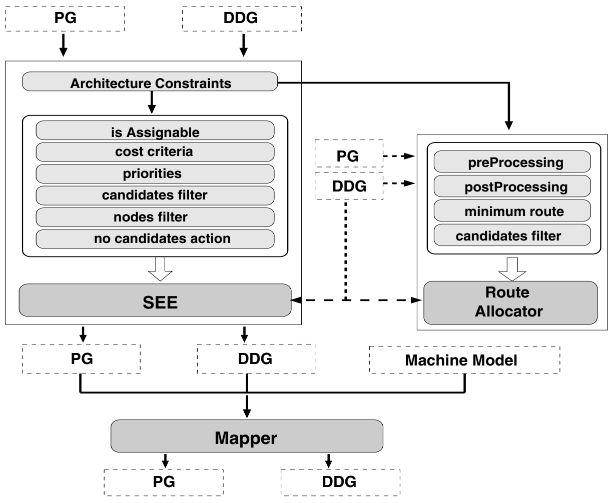


Figure 4. Cluster Assignment Framework

nodes of the PG, compatibly with the topology and the *Architecture Constraints*. These allow modeling typical constraints of the reconfigurable architecture, like the maximum number of input/output neighbors per cluster.

SEE is a local-scope based algorithm schema, which maintains a limited exploration frontier. It picks a new DDG node (or a set of nodes) at each step from a *priority* list of unassigned ones. For each node *c* of the PG, SEE checks if the current node n *isAssignable* to *c*, by taking into account the resource consumption and the availability of communication patterns. If so, *c* becomes a new candidate for n and the assignment $n \to c$ is evaluated by an objective function based on a collection of *cost criteria*. The list of candidates is then reduced by a *candidate filter*, the assignment $n \to c$ is performed and the partial solution is added as a new node to the exploration space frontier. In order to avoid an exponential explosion of the size of the frontier, the *node filter* prunes low-quality partial solutions. The concept is explained by Figure 5, where each node is a partial solution, and the move from a partial solution to another occurs assigning a node to a given candidate. The grey zone highlights the current frontier, which is kept of limited size by node filtering.

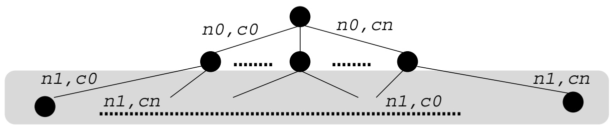


Figure 5. Space Exploration

It is sometimes impossible to find candidates. This situation occurs when *(i)* the *candidate filter* is too severe, i.e. all the available candidates are discarded because they strongly degrade the objective function value, *(ii)* there are no more communication patterns available. For instance, let us assume the following *isAssignable* interface implementation: cluster c is considered a valid candidate for n only if the already assigned neighbors of n can directly reach it. With respect to Figure 6 (a) no candidates are available for n, assuming the instantiated connections represented with continuous lines, the potential connections represented by dashed lines and the limit on input arcs per cluster set to 2. The assignment of n to C_0 or C_1 will cause the selection of an additional input line (to C_0 or C_1, respectively), which violates the constraints, while C_2 and C_3 cannot be considered candidates at all.

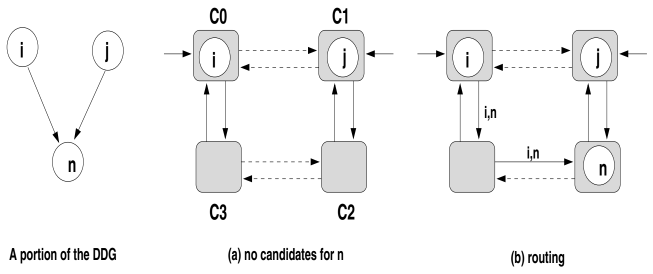


Figure 6. A routing example

When no candidates can be found a *no candidates action* is performed in order to escape from the empasse. A possible action can be the invocation of the configurable Route Allocator, which tries to assign the current DDG node to a convenient cluster, then routing the copies from/to its predecessors/successors as shown in Figure 6 (b), where available paths are used to route a copy from i to n passing through intermediate clusters.

The result of the cluster assignment is a completely assigned DDG and a PG reporting the copy flow over its arcs, now representing feasible communication patterns.

The last module, the Mapper, takes the assigned DDG, the PG and a complete description of the Machine Model as input, where all the communication wires are described at a low level of details. This module maps the PG onto the Machine Model, compatibly with available real communication paths and being driven by a configurable cost function, e.g. copy balancing, prioritization of parallel copies.

Figure 7 shows the differences between the PG and the Machine Model levels of abstraction. Four clusters connected by multiplexers can be seen, at a high level, as a complete graph, since each cluster can potentially reach all the others. The Mapper has to distribute the copies occurring in the PG onto the real communication wires of the MUXes.

This framework is currently a prototype, implemented in Python.

4 Hierarchical Cluster Assignment

Instruction Cluster Assignment (ICA) assigns each node of a DDG to a PE, trying to minimize a given cost function.

A way to attack the problem is to see it at a high abstraction level, where the machine topology is represented

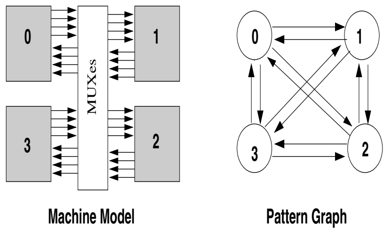


Figure 7. Example of Pattern Graph

by a graph of interconnected PEs and the only machine-dependent details needed are: *(i)* the resources requested by each node of the DDG, i.e. the functional units *(ii)* the latencies between adjacent nodes of the DDG, *(iii)* the resources available on each PE, *(iv)* the topology of the interconnection network.

The cost function may includes different parameters, some of them purely topological – e.g. number of intercluster copies, load balancing – and some other more architecture-oriented, which try to look forward to the following phases of the compilation flow – e.g. scheduling, register allocation.

When *(iv)* is a coarse-grain reconfigurable network with a limited number of selectable connections, then the problem grows in complexity, having to deal with a new group of constraints. Moreover, if the organization of the interconnection network is strongly hierarchical, as shown in Figure 2, it is hard even to abstract the whole topology as a graph or to represent it using sufficiently generic data structures. For instance, since each CN of Figure 2 can directly reach all the others, the whole topology could be seen as a K_{64} graph. But, in this case, it is necessary that the ICA keep trace of the internal logic of the hierarchy of MUXes, the paths between clusters are dependent on the current configuration of each MUX, and the number of parallel paths grows with the capacities of the MUXes as multiplication factors. With respect to Figure 2, two *computation nodes* at different sides of level 0 MUXes are potentially connected by $K^2M^2N^2$ parallel shortest paths.

Hence, we propose to exploit the hierarchical interconnection organization to decompose the ICA problem in multiple hierarchical algorithmic passes, each of them considering only one level of the hierarchy. In the remainder of this section we present the Hierarchical Cluster Assignment (HCA), i.e. a hierarchical decomposition of the ICA focusing on the interfaces between adjacent levels of the hierarchy.

A few details will be provided about the main cost factor we have considered in our objective function at each level of the hierarchy. In the following, we refer to the DSPFabric instance shown in Figure 2, where the architecture is homogeneous and each CN is a single-issue machine exposing the same set of resources – an ALU and an Address Generator(AG) to the DMA. Moreover, we refer when needed to the software modules introduced in the previous section.

4.1 Decomposition of the Problem

We decompose the ICA problem in multiple hierarchical ICA sub-problems, each of them addressing a specific level of the communication hierarchy. Each sub-problem is fully described by a DDG, a Working Set(WS), a constrained PG and an Inter Level Interface (ILI), and it is identified by a unique sequence of indexes, representative of its level of nesting as shown by Figure 8 (a). We will use that indexes in the following, in order to refer to the input data of each sub-problem.

In order to distinguish the data structures after ICA from themselves at the beginning of the algorithm we will use the over lined notation, e.g. $\overline{DDG}$ and $\overline{PG}$ are the DDG (the PG, respectively) after ICA. Moreover, we indicate with $\overline{DDG}(x)$ the cluster the instruction x is assigned to, and $\overline{PG}(c)$ the list of instructions assigned to cluster c. Finally, we label the arcs of $\overline{PG}$ with a description of the values they carry saying that $cpy(\overline{PG}(c,d))$ is the list of values on the arc from the cluster c to cluster d. We call these values inter-cluster *copies*.

The DDG represents the application to clusterize. The portion of the DDG to consider at a given level of the hierarchy is specified by the WS, in accord to $WS(DDG_{\dots,i,j}) = \{x \in DDG_{\dots,i,j} | \overline{DDG_{\dots,i}}(x) = j\}$ and to $WS(DDG_0) = \{x \in DDG_0\}$

Each node of the PG embraces a set of CNs, its resource table being the unitory of all the RTs of the CNs it includes. For instance, each node of PG_0 contains 16 ALUs/AGs, each node of $PG_{0,i}$ contains 4 ALUs/AGs and each node of $PG_{0,i,j}$ contains only one ALU/AG, $0 \leq i,j \leq 3$. Figure 8 explains the concept.

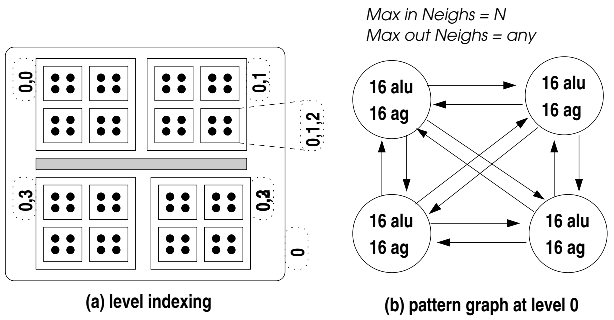
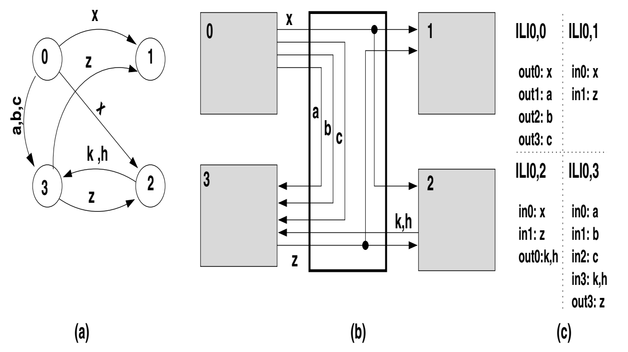


Figure 8. Problem decomposition

Each arc of the PG represents a *potential* communication pattern between two nodes and becomes a *real* communication pattern at the time an inter-cluster copy is effectively allocated by the algorithm onto it. The number of real communication patterns is limited by a group of constraints, which specifies the maximum number of input/output neighbors allowed for each node. The constraints must ensure that the module `Mapper` will be able to map $\overline{PG}$ onto the Machine Model. For instance, since the MUXes capacity at level 0 is N, we impose that the maximum number of input neighbors of a PG node do not exceed N. On the other hand, since it is possible to broadcast a value to more than one neighbor, we do not limit the number of output neighbors.

The HCA algorithm starts at level 0, mapping DDG_0 onto PG_0. Then the module Mapper maps $\overline{PG_0}$ onto the first level of the Machine Model Hierarchy, distributing the copies reported on the $\overline{PG_0}$ arcs over the physical communication wires and aiming at minimize the merit factor involved at the current level. Since our goal is to minimizes the II of the loop – as described in the next section – the Mapper tries to balance the copies in order to decrease their pressure on a single communication wire, as explained by Figure 9. $\overline{PG_0}$ is shown by (a): the value produced by instruction x is broadcasted from cluster 0 to clusters 1 and 2, meaning that instruction x of the DDG has been assigned to cluster 0 and some of its successors has been distributed over clusters 1 and 2. Another broadcasted instruction is z, while all the other copies do not need broadcast. Compatibly with the availability of communication wires, the Mapper uses only one line to broadcast x and z, moreover it tries to use all the possible communication patterns to map the remaining copies, e.g. distributing a, b and c over three wires, as highlighted by (b).



ILI0,0	ILI0,1
out0: x	in0: x
out1: a	in1: z
out2: b	
out3: c	

ILI0,2	ILI0,3
in0: x	in0: a
in1: z	in1: b
out0:k,h	in2: c
	in3: k,h
	out3: z

Figure 9. (a) PG after ICA. (b) Copy distribution. (c) ILIs to subproblems

The Mapper generates also four ILI ($ILI_{0,0}, \ldots ILI_{0,3}$), each of them reporting the input/output copies between level 0 and $0, i$ – $0 \leq i \leq 3$ – as shown by Figure 9 (c). For instance, $ILI_{0,3}$ reports that there are four input lines to the subproblem $0, 3$ carrying a, b, c and k, h, respectively, and one output line carrying z. More in general, the Mapper produces an ILI for each subproblem of the current one.

Now the communication paths at level 0 of the hierarchy has been allocated and the process can be iterated through all the nested levels, until a leaf problem is reached. Each subproblem loads the ILI interface given by its father and exploits the information about the incoming and outgoing connections for completing its PG with special nodes, which allow to maintain the communication flow between adjacent levels coherent.

A special *input node* is added for each incoming wire. It contains the list of copies pumped from the father into the current level, and it is connected by *potential* patterns to all the other nodes, meaning that the ingoing values can be broadcasted to all the clusters.

At the same manner, a special *output node* is added for each outgoing wire, containing the list of copies sent to the father. All the nodes are connected by *potential* patterns to all the output nodes and a new constraint is added to the problem, requiring that there must exist only one *real* pattern (only one arc) between a cluster and an output node – i.e. multiple clusters sending values on the same wire are avoided, in accord to the MUXes unary fan-in.

Figure 10 depicts the concept, considering the subproblem $0, 2$ and the $ILI_{0,2}$ of Figure 9. (a) shows a portion of the DDG, where the WS is highlighted in light-grey. Ingoing copies are x and z, which are incoming from two input wires, and outgoing copies are k, h on a single output wire. The PG completed with input and output nodes is drawn in (b), where it is also reported the additional constraint – here called *outNode_MaxIn* – which enforce to select only one arc to connect a cluster to an output node. (c) shows the PG after ICA, highlighting only the real patterns (those arcs that carry at least one copy). Let us notice that, in order to satisfy the additional constraint, both the instruction k and h has been assigned to the same cluster.

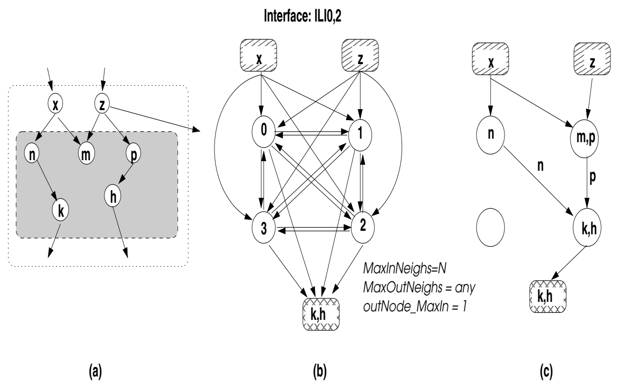


Figure 10. (a) A DDG portion, WS shown in grey. (b) PG completed with special nodes. (c) Real patterns after ICA.

Finally, when the Mapper has to deal with $\overline{PGs}$ including special nodes, as that one shown in (c), it must consider that there are incoming/outgoing connections from/to the outer level that cannot be used for copy distribution, partially limiting the reconfiguration space. These connections must be preallocated by the Mapper, being the glue between the outer and the inner level. Figure 11 shows the preallocated wires, w.r.t the $\overline{PG}$ of Figure 10 (c).

Working in this manner ensures that the copies flow balance at each level is kept coherent, guaranteeing the generation of legal code. The generation of good quality code is instead controlled by the cost functions involved at each subproblem.

At the end of HCA, a post processing pass exploits the informations held at the leaves of the problem hierarchy,

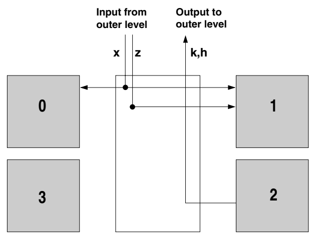


Figure 11. Preallocation of communication wires from/to the outer level.

in order to build the final DDG. Each DDG node is assigned to a CN and `receive` primitives are added as new DDG nodes, which perform the migration of the operands between different CNs.

All the connections are then allocated, producing the final topology of the architecture and a coherency checker verifies if the DDG is compatible with the topology itself. More precisely it checks for the presence of a communication paths on the final architecture between each pair of cluster that contains dependent nodes of the DDG.

4.2 Single Level Cluster Assignment

In our compilation flow, the cluster assignment phase will be followed by a modulo scheduling [20] pass on the clusterized DDG. Hence, the main cost factor we want to keep low in the objective function is the Minimum Initiation Interval of the loop (MII) [20] [21], computed as $max(iniMII, maxClsMII)$, where $iniMII$ is the MII at level 0 of HCA and $maxClsMII = max\{MII(x)|x \in PG_0\}$. The MII on a single cluster is computed by considering the maximum between the MII given by data constraints, $MIIRec$, and the MII given resource constraints $MIIRes$ [20], also taking into account a term of copy pressure computed considering the intercluster copies flowing on the PG arcs.

5 Experimental Results

In order to check the correctness of our HCA technique, we have clusterized four DDGs of significant loops of multimedia applications, i.e. `fir2dim`, `idcthor`, `mpeg2inter` and `h264deblocking`.

The first is a 2-dimensional fir filter, taken from the DspStone [26] benchsuite, the second is the Inverse Discrete Cosine Transform, taken from OpenDivx [16], the third is the interpolation filter of mpeg2 decoding algorithm and the latter is the row deblocking of h.264 algorithm.

The DDGs have been generated by an internal compiler front-end provided by STMicroelectronics. We have then used the HCA approach to clusterize them onto several instances of 64-clusters DSPFabric architecture, differing each from the other by the bandwidth parameters N, M and K shown by Figure 2. Table 1 reports the best results only, achieved for an architecture with $N = 8$, $M = 8$ and $K = 8$. What emerged during our experiments is that lower bandwidths cause a rapid degradation of the clusterization quality, since the interconnection network is not able to distribute the high number of intercluster copies, which are the main limiting factor to the final MII.

As the focus of this paper is neither to explore the architecture design space, e.g. tune the capacities of the MUXes, nor to optimize the heuristic pass involved at each non-hierarchical level, but we accurately checked that the HCA pass generates a legal clusterization, maintaing the coherence between adjacent levels. It is checked by the coherency checker at the end of HCA process, as described in previous section.

Hence, Table 1 focuses the attention on the legality of the final result, without listing the complete gamma of architecture exploration and heuristic tuning experiments we have performed. However, few words should be spent on the value of the MII we have achieved after clustering; as can be observed, this value is quite close to the theoretical optimum computed on an equivalent issue width unified bank machine. We remark that this value of MII is a lower bound for the following phase of modulo scheduling and we guess that it could increase dramatically unless we take into account scheduling aware cost factors – e.g. register pressure – at clustering phase. This will be part of planned future work, when we will implement the modulo scheduling phase, the register allocation and the DMA programming, the latter in order to keep the loop execution synchronous with the memory accesses. Moreover, we plan to test the code generated by the whole toolchain directly on an on-silicon prototype, which will be provided soon by STMicroelectronics.

Table 1. HCA test on four multimedia application loops

Loop	N_Instr	MIIRec	MIIRes	Legal clusterization	Final MII
fir2dim	57	3	2	yes	3
idcthor	82	1	2	yes	3
mpeg2inter	79	6	2	yes	8
h264deblocking	214	3	4	yes	6

6 Related Works

Reconfigurable loop accelerators such as GARP [10] and Morphosys [15] achieve good speedups with regard to general-purpose processor. Both are FPGA like with reconfigurable cells and mesh-based interconnections, but just the latter presents a fixed three level hierarchical interconnection network. Another mesh-based reconfigurable workstation is RAW [13]. It's realized by a mesh of modified MIPS with local caches and it provides both static and dynamic configurable communication network with variable delay.

Architectures based on linear arrays (RaPiD [9] and PipeRench [8]) are characterized by arrays of fixed size ALUS, immersed in a reconfigurable interconnect. They are designed aiming on mapping pipelines onto it.

PADDI-2 [25] shares several design aspects with DSP-Fabric, in particular way the two level hierarchical connections among PEs. Previous published works on RCP and DSPFabric are [6] [3].

Algorithms for instruction assignment on clustered architectures have been proposed by Desoli [7] and Lapinskii *et al.* [12] [11]. Chu *et al.* [4] proposed a hierarchical algorithm to iteratively partition a DDG on a clustered machine, but the target architecture considered by the authors is not hierarchical itself and is not reconfigurable.

Scheduling aware ICA approaches can be found in [2], [1] and [22]. Lee *et al.* has proposed integrated approach for clustering and scheduling on a Raw machine [13]. Another space-time scheduling approach can be found in [17].

Modulo scheduling aware ICA is presented by Nystrom and Eichenberger [18], while a unified distributed modulo scheduling algorithm is described in [5]. Convergent Scheduling [14] is an interactive and flexible framework for performing cluster assignment and scheduling over reconfigurable architectures. Finally, Modulo Graph Embedding [19] is a 3-D technique for performing space-time modulo scheduling for reconfigurable hardware. All the cited works do not handle hardware with hierarchical reconfigurable connections explicitly.

7 Conclusions and Future Works

In this paper we have introduced an innovative hierarchical cluster assignment framework. It is aimed at exploiting emergent massive parrallel clustered VLIW architectures with reconfigurable interconnection network. For large (say more than 32) hierarchical clusters network reconfigurability causes an explosion of the number of feasible topologies. To make this problem tractable, we have carefully designed a novel decomposition approach, which considerably cuts the state-space exploration of subproblems in accord with the hierarchy of the interconnections. On going research aims at tuning of the heuristics and cost functions used to perform cluster allocation in order to take into account more scheduling aware parameters, e.g. the register pressure.

References

[1] C. Akturan and M. F. Jacome. Caliber: A software pipelining algorithm for clustered embedded VLIW processors. In *Proc. ICCAD*, pages 112–118, 2001.

[2] A. Aleta, J. Codina, J. Sanchez, and A. Gonzalez. Graph partitioning based instruction scheduling for clustered processors, 2001.

[3] J. Cambonie. A hierarchical reconfigurable computer architecture. Patent (application number 05112850).

[4] M. Chu, K. Fan, and S. Mahlke. Region-based hierarchical operation partitioning for multicluster processors. In *Proc PLDI '03*, pages 300–311, New York, NY, USA, 2003. ACM Press.

[5] J. M. Codina, J. Snchez, and A. Gonzlez. Uracam: A unified register allocation, cluster assignment and modulo scheduling approach. 2001.

[6] O. Colavin and D. Rizzo. A scalable wide-issue clustered vliw with a reconfigurable interconnect. In *CASES '03*, pages 148–158, New York, NY, USA, 2003. ACM Press.

[7] G. Desoli. Instruction assignment for clustered vliw dsp compilers: A new approach. Technical Report HPL-98-13, Hewlett-Packard, Feb 1998.

[8] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. Piperench: A coprocessor for streaming multimedia acceleration. In *Proc. ISCA*, pages 28–39, 1999.

[9] C. E. D. C. Green and P. Franklin. RaPiD – reconfigurable pipelined datapath. In R. W. Hartenstein and M. Glesner, editors, *Field-Programmable Logic: Smart Applications, New Paradigms, and Compilers. 6th International Workshop on Field-Programmable Logic and Applications*, pages 126–135, Darmstadt, Germany, 1996. Springer-Verlag.

[10] J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In K. L. Pocek and J. Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 12–21, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[11] V. Lapinskii, M. Jacome, and G. de Veciana. Application-specific clustered vliw datapaths: Early exploration 32 on a parameterized design space, 2002.

[12] V. S. Lapinskii, M. F. Jacome, and G. de Veciana. Cluster assignment for high-performance embedded vliw processors. *ACM Trans. Design Autom. Electr. Syst.*, 7(3):430–454, 2002.

[13] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. P. Amarasinghe. Space-time scheduling of instruction-level parallelism on a raw machine. In *Proc. ASPLOS*, pages 46–57, 1998.

[14] W. Lee, D. Puppin, S. Swenson, and S. Amarasinghe. Convergent scheduling. In *MICRO 35*, pages 111–122, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[15] G. Lu, H. Singh, M.-H. Lee, N. Bagherzadeh, F. J. Kurdahi, and E. M. C. Filho. The morphosys parallel reconfigurable system. In *European Conference on Parallel Processing*, pages 727–734, 1999.

[16] P. Mayo. Opendivx. http://www.projectmayo.com.

[17] R. Nagpal and Y. N. Srikant. Integrated temporal and spatial scheduling for extended operand clustered vliw processors. In *Proc. CF '04*, pages 457–470, New York, NY, USA, 2004. ACM Press.

[18] E. Nystrom and A. E. Eichenberger. Effective cluster assignment for modulo scheduling. In *International Symposium on Microarchitecture*, pages 103–114, 1998.

[19] H. Park, K. Fan, M. Kudlur, and S. Mahlke. Modulo graph embedding: Mapping applications onto coarse-grained reconfigurable architectures, 2006.

[20] B. R. Rau. Iterative modulo scheduling: an algorithm for software pipelining loops. In *MICRO 27*, pages 63–74, New York, NY, USA, 1994. ACM Press.

[21] B. R. Rau, M. S. Schlansker, and P. P. Tirumalai. Code generation schemas for modulo scheduled loops. In *25th Annual International Symposium on Microarchitecture*, Portland, Oregon, 1992. IEEE and ACM.

[22] F. J. Sánchez and A. González. Instruction scheduling for clustered vliw architectures. In *ISSS*, pages 41–46, 2000.

[23] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal. Scalar operand networks: Design, implementation, and analysis.

[24] A. Yeung and J. Rabaey. A reconfigurable data-driven multiprocessor ic for rapid prototyping of high performance dsp algorithms. In *Proc HICSS-26*, Kauai, Hawaii, Jan 1993.

[25] A. K. W. Yeung and J. Rabaey. A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput dsp algorithms, 1993.

[26] V. Zivojnovic, J. M. Velarde, C. Schläger, and H. Meyr. DSPstone–A DSP-oriented Benchmarking Methodology. *ICSPAT*, 1994.