Heterogeneous Clustered VLIW Microarchitectures

Àlex Aletà[1], Josep M. Codina[1,2], Antonio González[1,2], David Kaeli[3]
1. Dept. of Computer Architecture, UPC, Barcelona, Spain
2. Intel Barcelona Research Center, Intel Labs, UPC, Barcelona, Spain
3. Northeastern University, Boston (MA)
aaleta@ac.upc.edu, josep.m.codina@intel.com, antonio.gonzalez@intel.com, kaeli@ece.neu.edu

Abstract

Increasing performance, while at the same time reducing power consumption, is a major design tradeoff in current microprocessors. In this paper, we investigate the potential of using a heterogeneous clustered VLIW microarchitecture. In the proposed microarchitecture, each cluster, the interconnection network and the supporting memory hierarchy can run at different frequencies and voltages. Some of the clusters can then be configured to be performance-oriented and run at high frequency, while the other clusters can be configured to be low-power-oriented and run at lower frequencies, thus reducing overall consumption.

For this heterogeneous design to be effective, we need to select the most suitable frequencies and voltages for each component. We propose a scheme to choose these parameters based on a model that estimates the energy consumption and the execution time of floating-point codes at compile time.

Finally, we present a modulo scheduling technique based on graph partitioning that exploits the opportunities presented on heterogeneous clustered microarchitectures.

Results show that the Energy-Delay^2 product (ED2) can be significantly reduced by 15% on average for a microarchitecture with 4-clusters and by as much as 35% for selected programs.

1. Introduction

Power consumption is becoming a major issue in the design of current microprocessors. Statically-scheduled processors have been shown to be an attractive design point when trying to reduce the power budget. Clustering is becoming a common trend in the design of current microprocessors due to its ability to mitigate wire delays, and reduce both power dissipation and complexity. Therefore, clustered statically-scheduled processors are a good alternative when considering the power/performance tradeoff. This design has become quite visible in the DSP market [14][10][13][26].

Clustering consists of partitioning processor resources into several groups or *clusters*. The components of each cluster are simpler, faster, and consume less power. The resources in a cluster can be laid out in close proximity, which reduces signal transmission delays [1]. Long (and slow) wires are used to interconnect clusters.

In this paper, a new microarchitecture and a set of compilation techniques that can enhance statically-scheduled clustered designs are presented. The proposed enhancements pursue high-performance while reducing power consumption. The key feature of the proposed microarchitecture is that it allows for the use of different frequencies and voltages in each component. We refer to this design as a *heterogeneous* microarchitecture, as opposed to traditional *homogeneous* designs where the whole processor is working at the same frequency and voltage.

A heterogeneous clustered model is designed to take advantage of the fact that in most applications, a small subset of the instructions is critical for performance. These instructions can be scheduled in performance-oriented clusters that are clocked at a higher frequencies (using higher voltages at the expense of power), whereas the remaining instructions are placed in low-power-oriented clusters running at lower frequencies (hence using lower voltages and consuming less power).

A model is presented in this paper that computes at compile time an accurate estimate of the energy consumption, as well as the execution time for a particular application, as run on a particular heterogeneous configuration. Using this model, efficient heuristics have been defined for selecting the most suitable frequencies and voltages for each

International Symposium on Code Generation and Optimization (CGO'07)
0-7695-2764-7/07 $20.00 © 2007 IEEE

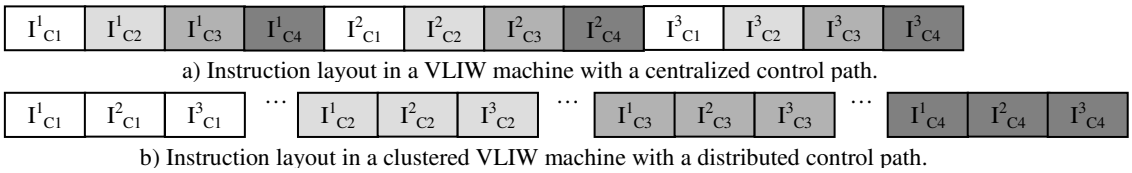


Figure 1: Code layout in clustered VLIW machines.

component in order to maximize performance and minimize energy consumption.

The effectiveness of a clustered statically-scheduled processor depends heavily on the compiler. Among the different compiler steps, one of the most critical is code scheduling. During this step, the compiler is also responsible for performing cluster assignment.

In this paper, we consider how best to perform instruction scheduling for heterogenous clustered VLIW microprocessors. In particular, we focus on software-pipelined loops [6] since they account for most of the execution in floating point code. We have developed a modulo scheduling algorithm for heterogeneous microarchitectures that strives to improve the energy-delay^2 product (ED2).

Our simulation results show that heterogeneous microarchitectures can produce substantial benefits in terms of ED2 when frequencies and voltages are selected according to the proposed scheme and the described modulo scheduling technique is used. In particular, the heterogeneous design is able to achieve on average a 15% benefit in ED2 and up to 35% for selected programs.

The rest of this paper is organized as follows. Section 2 provides the background on the proposed heterogeneous scheme, including the definition of the microarchitecture and the extensions considered for modulo scheduling. Section 3 describes the technique used for selecting frequency and voltages. Section 4 presents our novel modulo scheduling technique. Section 5 evaluates the the design space. Section 6 reviews related work and section 7 summarizes the paper.

2. Basics of Heterogeneous Design

2.1 Description of the Microarchitecture

In this work, we focus on statically-scheduled clustered microarchitectures that use a distributed control path [32]. Each cluster is a semi-independent unit composed of FUs, memory ports and a register file. Clusters communicate register values among them using special copy instructions and a set of dedicated *register buses*.

The fetch and decode units are also distributed across clusters. Hence, instructions assigned to the same cluster are laid out together, as shown in Figure 1. Therefore, when a branch instruction occurs, each cluster has a different destination, and so all clusters must have their own PC and the logic necessary to update it. Branches are handled according to the unbundled branch architecture specified by HPL-PD [18], i.e., they are decoupled in several instructions:

1. Branch target computation: in each cluster, an instruction specifies the branch destination.
2. Branch condition evaluation: the branch has to be evaluated in one of the clusters only. Then, its result is broadcast to the remaining clusters.
3. Control transfer: if the branch is taken, a specific instruction will transfer control to the appropriate destination in each cluster.

Traditionally, the same frequency and voltage are used across the entire microprocessor design. This scheme is refered as the homogeneous microarchitecture. In this work we present a *heterogeneous* microarchitecture. This scheme allows for voltage and frequency scaling in different components of the processor. The proposed microarchitecture will be organized as a Multi-Clock Domain (MCD) processor [30]. The different clock domains are stablished following the natural division of the microarchitecture. Since clusters work almost independently, they are defined as a domain. The interconnection network and the memory hierarchy are configured as domains too. Different domains are synchronized by using queues, as shown in Figure 2. These queues often introduce delays of one cycle due to syncronization problems.

In order to allow for frequency scaling, dedicated logic is needed in the hetereogenous system to generate the appropriate clock signals and distribute them to each component. As shown in Figure 2, a limited number of possible frequencies are generated from a general clock signal (the *gen_clock* signal in Figure 2) using multipliers and dividers. Then, the appropriate frequency is selected using a multiplexor.

The clock generation network can either be placed on-chip (using delay-locked-loops [25][12]), off-chip (using phaselocked loops), or using a combination of both approaches. For simplicity purposes, we will assume that this hardware is placed on chip.

When different frequencies are used in different components of an microarchitecture, it sometimes

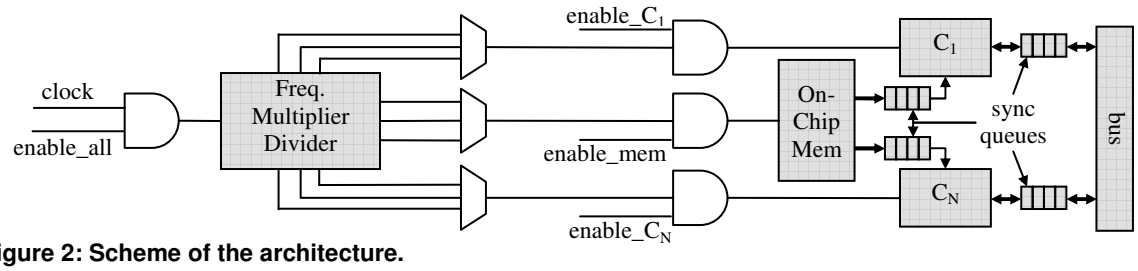


Figure 2: Scheme of the architecture.

becomes necessary to synchronize them (i.e., all clock periods starting simultaneously). A good example of this is at the beginning of a loop execution. To achieve synchronization, a set of enable signals are used (Figure 2). Initially, all enable signals are set to zero so that no clock signal is sent to the heterogeneous components. On the next general clock signal, the *enable_all* signal is set to one. Hence, a synchronized signal is produced for each component. This signal takes one cycle to reach the different components. During this cycle, the signal is unstable, so individual enables (i.e. *enable_memory*, *enable_C1* and *enable_CN* in Figure 2) are kept at zero. After waiting one cycle, individual enables are set to one so that the appropriate signal is received at each component.

On the other hand, voltage scaling is done by means of voltage regulators. These can also be internal or external to the chip. Voltage scaling takes longer than frequency scaling. However, since the reconfiguation in our approach is only performed at a program level, voltage changes have a negligible impact on performance.

In order to exploit the potential of this microarchitecture, a novel modulo scheduling algorithm for heterogenous configurations is presented. For this purpose, some extensions to the traditional definition of the modulo scheduling framework described in [27] are proposed. Background on modulo scheduling, as well as ourextensions, are introduced in the next section.

2.2 Modulo Scheduling for Heterogeneous Microarchitectures

The applications executed in statically scheduled microarchitectures typically spend most of their execution time in loop bodies. For this reason, in this work we present code generation schemes that specifically target scheduling loop bodies. In particular, we will describe a technique to apply modulo scheduling to heterogeneous clustered microarchitectures.

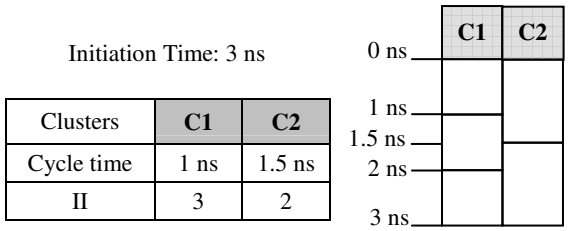
Modulo scheduling (*MS*) is a well-known software pipelining technique [27]. In a modulo scheduled loop, a new iteration starts before the previous one finshes. The number of cycles between the start of two consecutive iterations of a given loop is a constant, and is called the *initiation interval* (*II*). The *II* is smaller than the number of cycles that a single iteration takes to complete its execution, which will be refered to as the *iteration length* (*it_length*). Hence, there are instructions from different iterations executing at the same time. The maximum number of iterations that are executed concurrently is referred to as the *stage count* (*SC*). These factors are related to the total execution time of the scheduled loop as follows:

$$T_{exec} = (N-1+SC)*II* T_{cyc}$$
$$SC = \lceil it\_length/II \rceil$$

where T_{cyc} stands for the cycle time of the processor and *N* is the number of iterations of the loop. Thus, when *N* is large, the execution time of a modulo scheduled loop is almost proportional to the *II*.

The structure of the *data dependence graph* (*DDG*) of the loop and the microarchitecture of the target machines impose some constraints on the *II*. The *minimum initiation interval* (*MII*) is a minimum bound of the *II*. It can be computed as the maximum between the *recurrence minimum initiation interval* (*recMII*), that takes into account recurrence circuits in the DDG, and the *resource minimum initiation interval* (*resMII*), which takes into account the workload and the available resources in the processor (*resMII*):

Nevertheless, when targeting heterogeneous microarchitectures, the basic modulo scheduling equations presented above must be modified. The elapsed-time between the start of two consecutive iterations must remain constant for a loop (else we would have synchronization problems occurring across iterations). We will refer to this time as the *initiation time* (*IT*). However, the number of cycles between the start of two consecutive iterations may differ depending on the component of the microarchitecture that is being considered, because different components may run at different frequencies. Therefore, in a heterogeneous microarchitecture, the *II* is no longer a constant across the microprocessor. An example of the characteristics of modulo-scheduling for heterogeneous microarchitectures is shown in Figure 3. In this example, a 2-clustered machine is assumed. The first cluster runs at 1ns whereas the second runs at 1.5ns. If a loop iterates every 3 ns(*i.e.*, *IT*=3 ns), the *II* for cluster 1 is 3 cycles, but for cluster 2 the *II* is 2 cycles.



Clusters	C1	C2
Cycle time	1 ns	1.5 ns
II	3	2

Figure 3: The IT for the heterogeneous and the II and the frequency for each cluster.

On the right side of the figure the relation between the *II*s is specified.

For a given component *X*,we will use II_X to denote the initiation interval of this component. The following equation relates *IT* to the II_X:

$$II_X = IT \cdot f_X$$

where f_X stands for the frequency of component *X*. Thus, instead of having a constant *II* for the whole processor, we will have a set of pairs (frequency, *II*), with one pair for each component of the microarchitecture running at that different frequency.

Similarly, the definition of the *MII* is extended to the *minimum initiation time* (*MIT*), which will be the maximum of either the *recurrence minimum initiation time* (*recMIT*) or the *minimum initation time* (*resMIT*). The *recMIT* takes into account recurrence circuits in the *DDG* of the loop, and is equal to the *recMII* multiplied by the cycle time of the fastest cluster in the microarchitecture:

$$recMIT = recMII \cdot \min_{C_i \text{ cluster}} \{T_{cyc_{C_i}}\}$$

In Figure 4 an example is presented. To the left, the *DDG* of a loop is shown. Instructions A, B and C form a recurrence. We assume a 2-clustered microarchitecture. The fastest cluster is C1 which runs at 1ns. If all instructions have 1 cycle latency, the *recMIT* would be *3 cycles x 1 ns/cycle = 1ns*.

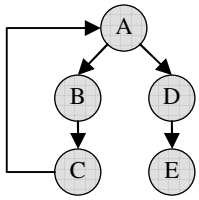
The *resMIT* is computed by taking into account (using the different *IIs* obtained for each cluster) that there must be sufficient slots to schedule all the instructions of the loop. For the example in Figure 4, we present a table containing the different *II*'s of each cluster for different values of the *IT*. Note that in order to schedule the five instructions of the *DDG*, we need an *IT*= 3.333 ns. With this *IT*,we can have three slots in *C1* and two slots in *C2*. As explained before, frequency scaling will be required in *C1*.

Later in Section 4 we will present an algorithm to perform modulo scheduling for heterogeneous architecutures which employs the concepts introduced in this section.

3. Selecting Voltages and Frequencies

In this paper, we are trying to design a heterogeneous microarchitecture whose different components can run at different frequencies and voltages in order to achieve high performance and, at the same time, redue energy consumption. In particular, the components that will use different voltages and frequencies are the clusters, the intercluster connection network, and the on-chip memory hierarchy. Some of the clusters will be performance-oriented and will run at higher frequencies using higher voltages. The reamining clusters will be low-power-oriented and will run at a lower frequencies, thus consuming less power. In order to achieve high performance, those instructions that severely impact the overall execution time will be scheduled in performance-oriented clusters. The remaining instructions will be scheduled in the slower, low-power-oriented clusters. The intuition behind this scheme is that only a small subset of the instructions is critical for performance and thus must be executed as fast as possible, while others can be delayed without significantly increasing the execution time.

In order to effectively exploit a heterogeneous design, a scheme to decide which frequencies and voltages should be applied to each component of the microarchitecture is required. Scaling voltages and frequencies can have a large impact on the execution time and on energy consumption. Hence, we need to estimate at compile time the effects that modifying these parameters will have on power/performance in order to appropriately configure the heterogenous design. For this purpose, we will first simulate program



Architecture	C1	C2
Cycle Time	1ns	1.67ns

recMIT	Recurrence {A,B,C}: 3 cycles **recMIT** = 3x1ns = 3 ns

resMIT:	IT	IIC1	IIC2	Capacity
5 instructions	1	1	0	1 slot
	1.667	1	1	2 slots
	2	2	1	2 slots
	3	3	1	4 slots
	3.33	3	2	**5** slots

MIT
max{resMIT,recMIT} =
=max{3.33ns; 3ns} =
MIT= 3.33ns

Figure 4: Computing the MIT for heterogeneous configurations.

execution in a reference homogeneous microarchitecture. From this execution we obtain profile data, that includes power consumption measurements and loop-related dynamic information (i.e., the average number of iterations). Using this profile information, we estimate the power consumption and the execution time of a heterogenous configuration in terms of our reference homogeneous microarchitecture. In the next subsections, the models used for this purpose are described.

3.1 Model for Estimating Energy Consumption

Our energy consumption model is based on the power consumption of a reference homogeneous microarchitecture First, we develop a formula to express the energy consumption of the reference model. For this purpose, we divide the energy consumption of the processor into three main factors: 1) energy consumed by the clusters, 2) energy consumed by the intercluster connection network, and 3) energy consumed by the memory hierarchy. In turn, these can be divided into dynamic and static subfactors. We can compute separately the six components of the total energy consumption. We begin with the static energy consumption. The leakage of the clusters is the sum of the leakage of each individual cluster. The leakage of a given component can be aproximated as the total execution time, multiplied by the average static energy consumption of that component in one second.

$$E_{stat_{CPU}} = T_{exec} \cdot \sum_{C_i} E_{s_{Ci}}$$

$$E_{stat_{ICN}} = T_{exec} \cdot E_{s_{ICN}}$$

$$E_{stat_{cache}} = T_{exec} \cdot E_{s_{cache}}$$

where $E_{s_{Ci}}$, $E_{s_{ICN}}$, $E_{s_{cache}}$ stand for the average static energy consumption during one second of cluster C, the cache, and the intercluster connection network, respectively.

For computing dynamic energy consumption, each the three components can be approximated as the total number of executed instructions/communications/ memory accesses, multiplied by the average consumption of one instruction/communication/access:

$$E_{dyn_{CPU}} = nIns \cdot E_{ins}$$

$$E_{dyn_{ICN}} = nComms \cdot E_{comm}$$

$$E_{dyn_{cache}} = nMemIns \cdot E_{access}$$

where E_{ins} stands for the average energy consumption of one instruction in a cluster, E_{comm} represents the average energy consumed by the bus during one communication, and E_{access} stands for the average energy consumed by the cache for one access. This model can be enhanced by dividing the instructions executed in the clusters into classes and, for each class, assigning the appropriate energy consumption. However, for the sake of simplicity, in this section we will assume that all of the instructions consume the same energy.

By combining these factors, the total energy consumed by the reference homogeneous clustered microarchitecture can be sumarized as follows:

$$E_{\text{hom}} = nIns \cdot E_{ins} + nComms \cdot E_{comm} + nMemIns \cdot E_{access} + T_{exec_{\text{hom}}} \cdot \left(nc \cdot E_{s_C} + E_{s_{ICN}} + E_{s_{cache}}\right)$$

In order to rewrite the above equation for the case of an heterogenous microarchitecture (where different voltages and frequencies are used for each component), we will compute the variations produced by frequency and voltage scaling in the six energy components previously used (*i.e.* $E_{ins}, E_{comm}, E_{access}$, $E_{s_{Cx}}, E_{s_{ICN}}, E_{s_{cache}}$). This is done in the following sections. We highlight two cases: 1) variations for dynamic energy units, and 2) variations for static energy units. We can write out a formula to compute the energy consumption of an arbitrary microarchitecture in terms of the reference homogeneous microarchitecture.

3.1.1 Variations in Dynamic Energy when Scaling Voltage and Frequency

Dynamic energy is related to the supply voltage and frequency using the following formulae:

$$E_{dyn} = P_{dyn} \cdot T$$

$$P_{dyn} = p_t \cdot f \cdot C_L \cdot {V_{dd}}^2$$

$$\Rightarrow E_{dyn}(1cyc) = p_t \cdot C_L \cdot {V_{dd}}^2$$

If the same instruction is to be executed in two clusters with the same design, but the two clusters use different voltages and frequencies, the instruction will still use the same number of cycles (though different time). Hence, the relationship between the energy consumed in each cluster is:

$$\frac{E_{dyn}}{E_{dyn_0}} = \frac{p_t \cdot C_L \cdot {V_{dd}}^2}{p_t \cdot C_L \cdot {V_{dd_0}}^2} = \left(\frac{V_{dd}}{V_{dd_0}}\right)^2 =: \delta$$

3.1.2 Variations in Static Energy when Scaling Voltage and Frequency

Static energy is related to supply voltage and threshold voltage as follows:

$$E_{stat} = P_{stat} \cdot T$$

$$P_{stat} = \frac{I_0}{W_0} W_t \cdot 10^{-V_{th}/S} \cdot V_{dd}$$

Hence, the relationship between the average static energy consumed during one second for two components with the same design, but using different voltages and frequencies, is as follows:

$$\frac{E_{stat}}{E_{stat_0}} = 10^{\frac{V_{th_0} - V_{th}}{S}} \cdot \frac{V_{dd}}{V_{dd_0}} =: \sigma$$

3.1.3 Variations of the Energy Consumption in the Microarchitecture

Given the previous formulae, we can express the total energy consumed by an arbitrary microarchitecture in terms of the energy consumption units of the reference homogeneous microarchitecture:

$$E_{het} = nIns \cdot E_{ins} \cdot \sum_{C_i} p_{C_i} \cdot \delta_{C_i} + nComms E_{comm} +$$

$$+ nMemIns E_{access} \cdot \delta_{cache} +$$

$$+ T_{exec_{het}} \cdot \left(E_{s_C} \cdot \sum_{C_i} \sigma_{C_i} + E_{s_{ICN}} \cdot \sigma_{ICN} + E_{s_{cache}} \cdot \sigma_{cache} \right)$$

where p_{C_i} stands for the probability that an instruction is executed in cluster C_i.

We can now estimate the energy consumed by any heterogeneous configuration with respect to the reference homogeneous microarchitecture. Moreover, this formula permits us to estimate the energy consumed by other homogeneous microarchitectures, using frequencies and voltages different than the reference homogeneous design.

3.2 Estimation of the Execution Time of an Heterogeneous Microarchitecture

In order to appropriately select the frequencies and voltages for each component of an heterogenous microarchitecture, we also need to estimate the effects that using an heterogeneous configuration will have on execution time.

For approximating the execution time of a modulo scheduled loop, we need to estimate two parameters of the final schedule: the *IT* of the loop and the time an iteration takes to be completed (*it_length*). To compute these values, we will use profiling information taken from the reference homogeneous microarchitecture.

The proposed technique estimates the *IT* of a loop in a particular heterogeneous configuration as the minimum time, such that:

the *IT* is greater than or equal to the *MIT* of the current heterogeneous design (i.e., there are enough slots to schedule all the instructions of the loop, and there is enough room to schedule the longest recurrence of the loop).

there are enough slots to accommodate the number of communications required by the schedule of the homogeneous microarchitecture.

there are enough lifetime slots to accommodate the sum of the lifetimes of all the values for the schedule of the homogeneous microarchitecture.

It is difficult to estimate the time that one single iteration takes to complete. However, this parameter has a mild impact on the execution time of modulo scheduled loops (especially if the number of iterations is high). In order to reasonably approximate this parameter, we assume that half the iteration will be executed on the performance-oriented clusters (i.e., fast clusters) and the remaining half in the low-power-oriented clusters (i.e., slow clusters). Based on this assumption, we can estimate the amount of time that an iteration takes to complete by multiplying the number of cycles that a single iteration requires in the homogeneous microarchitecture by the arithmetic mean of the cycle time of the heterogeneous clusters.

Using this modeling process, we can estimate the execution time of any particular heterogeneous configuration with respect to the reference homogeneous microarchitecture. When we combine this with the model that estimates the energy consumption of the heterogeneous microarchitecture, we can estimate ED2 and then choose the voltage and frequency that, according to these estimates, providethe greatest benefit.

3.3 Estimation of ED2 for the Heterogeneous Microarchitecture

Based on the formulae presented in the previous sections, and using the profiling information obtained from the reference homogeneous microarchitecture, it is possible to anticipate which will be the best heterogeneous configuration in terms of ED2.

We explore the different design alternatives: varying the number of fast clusters, the relative delay between the fast and the slow clusters and the supply voltages for each component. Given a frequency and a supply voltage,the threshold voltage is given by the following α power model formula:

$$f_{\max} = \frac{(V_{dd} - V_{th})^{\alpha}}{\beta \cdot C_L \cdot V_{dd}}$$

where f_{max} stands for the maximum frequency that this component can use, V_{dd} stands for the voltage suply, V_{th} for the voltage threshold, β is a constant specific to the technology, C_L stands for the capacitance and the α power reflects the fact that the transistors may be velocity saturated.

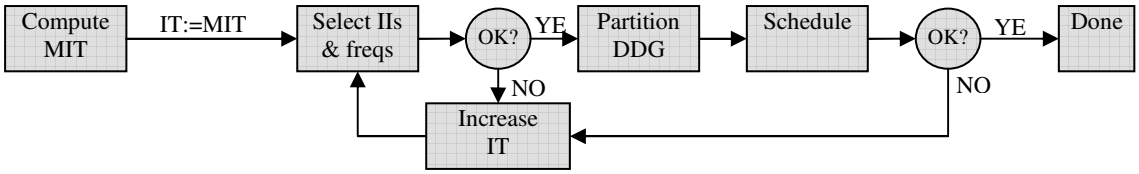


Figure 5: Overview of the proposed modulo scheduling algorithm.

The value obtained for V_{th} must satisfy the following equation in order to prevent metastability in sequential circuits and glitches in combinational circuits, and to deal with process variations, which may lead to V_{th} variations:

$$(V_{dd} - V_{th}) - V_{th} < 0.1 \cdot V_{dd}$$

For each different heterogeneous design, we estimate ED2 with the formulae as described in the previous sections. The design that produces the lowest ED2 estimation is selected.

4. Proposed MS Scheme

In this work we propose an algorithm to modulo schedule loops for heterogeneous clustered microarchitectures. The proposed scheme is based on a state-of-the-art work targeting homogeneous microarchitectures [2][3]. This code generation scheme targets an heterogenous design where voltages and frequencies have been selected beforehand, based on the approach presented in the previous section. These voltages determine a maximum frequency $f_{\max}$ for each component of the microarchitecture.

In Figure 5, we show the flow of the proposed algorithm. First, the *minimum initiation time* (*MIT*) is computed. Then, the *IT* is set equal to the *MIT*. Since there can be components of the microarchitecture running at different frequencies, a suitable *II* is found for each component running at a different frequency. In particular, for a given component *X* which runs at a voltage that determines a maximum frequency of $f_{\max_X}$ we have that:

$$II_x = \left\lfloor f_{\max_X} \cdot IT \right\rfloor$$

Thus, this component will have to run at a frequency :

$$f_x = II_X / IT \leq f_{\max_X}$$

Frequency scaling may be required every time a new loop is to be executed. Otherwise, we cannot allow for every possible frequency. For a given voltage, each component will be able to run at a limited number of frequencies. Sometimes we may not be able to find an appropiate pair (frequency, *II*) for a given *IT* and a given component. When this happens, we need to increase the *IT*. In such cases, we say that we increase the *IT* due to syncronization problems.

Once we have selected the appropriate pairs (frequency, *II*) for all of the components of the microarchitecture, the *DDG* of the loop is partitioned (that is, each instruction is assigned to a cluster) and is scheduled. If a suitable schedule can not be found, the *IT* is increased and the process starts again from selecting appropiate pairs (frequency, *II*).

In the next section we describe our graph partitioning scheme for heterogeneous designs.

4.1 Graph Partitioning

Graph partitioning is an NP-complete problem that has been thoroughly studied. Many heuristics have been proposed. Among them, multilevel strategies [17] have been shown to be particularly useful.

Multilevel strategies consist of two steps. During step one (referred to as coarsening), pairs of nodes of the original dependence graph are fused into new, coarser macronodes. Hence, a new, coarser graph is obtained with fewer nodes and edges. Each node of the orginal, finer, graph corresponds to one and only one node of the new, coarser graph. Thus, a partition of the new, coarser, graph induces a partition in the original graph. This process is iterated upon until a final (i.e., coarsest) graph is arrived at. This final graph will contain as many nodes as there are resources. Then, each node of this final (i.e., coarsest) graph is assigned to a different set of the partition. A preliminary partition is obtained, which in turn induces a partition in all the finer-grained graphs generated during the coarsening process.

The second step of multilevel strategies (referred to as refinement), attempts to improve the preliminary paritition that results from the coarsening process. Beginning with the coarsest graph and proceeding up to the original graph, heuristics are applied. These heuristics move nodes (at different coarse levels, depending on the graph to which they are being applied) from one set of nodes in the partition to another. At each step, the movement that produces the greatest benefit in terms of some metric is selected, and then the algorithm proceeds with the new partition until no further improvement can be found. Note that when a new partition is selected, this new partition induces a new partition in all of the finer graphs. Thus, the partition is improved up to the original graph.

Next, we describe how we have adapted a multilevel strategy for partitioning a *DDG* representing a loop in order to perform cluster assignment for MS for heterogeneous microarchitectures. The goal of the proposed technique is to produce a partition which can be scheduled to minimize ED2. For this purpose, we present heuristics that try to assign to the fast clusters only those instructions that are critical for execution time. The remaining instructions will be placed in slower clusters in order to reduce power consumption.

The proposed scheme is based on a technique oriented toward MS for homogeneous microarchitectures [2][3]. In the following section we will describe the enhancements proposed that specifically optimize heterogeneous microarchitectures. For a more detailed description of the algorithm used for homogeneous microarchitectures, the interested reader is referred to [2][3].

4.1.1 Coarsening for Heterogeneous Microarchitectures.

Coarsening schemes such as the algorithms described in [2][3] assume that all the sets of the partition have the same constraints. However, when targeting an heterogeneous microarchitecture, this assumption is not longer true. The frequency and the voltage of each cluster in an hetereogenous configuration may be different. This fact should be taken into account when partitioning the data dependence graph. In particular, heterogeneity can have a significant impact on scheduling recurrences. If a recurrence possessing a large latency is placed in a cluster with a low frequency, the *IT* will have to be increased. For that reason, in our algorithm, nodes in recurrences are treated differently.

Prior to the start of the coarsening process, we take care of the recurrences whose latency is larger than the *II* of any of the clusters. That means that there may be certain clusters where these recurrences cannot be scheduled for the current *IT*. We identify these recurrences and order them from the most critical (in terms of latency) to the least critical. Then, starting with the most critical, we place them, one at a time, in the slowest cluster where they can be scheduled. By proceeding in this way, we ensure, on one hand, that all recurrences are placed in clusters where they can be scheduled for the current *IT*. On the other hand, we are trying to keep energy consumption as low as possible (slower clusters consume less power). Note that with the described scheme, recurrences are not partitioned during this step. This is due to the fact that partitioning a critical recurrence is rarely beneficial because its latency increases considerabley due to two reasons. First, if an instruction contained in the recurrence is placed in a slower cluster, it takes longer to complete.

ISA	INT		FP	
	Lat	E	Lat	E
Memory	2	1	2	1
Arithmetic	1	1	3	1.2
Multiply	2	1.1	6	1.5
Division/Modulo/sqrt	6	1.4	18	2

Table 1: Latency of the instructions and energy consumption relative to an integer add.

Second, whenever a recurrence is partitioned across multiple clusters, intercluster communications are required. Thus, recurrences have not been partitioned when we arrive at a set of preliminary partitions, though if necessary, recurrences can be partitioned during the refinement step.

Once this subset of recurrences has been placed in suitable clusters. we proceed with coarsening.

4.1.2 Refining the Partition

Once the preliminary partition has been generated, we try to improve upon it by applying two different heuristics. The first tries to keep the partition *balanced*. A balanced partition refers to a partition such that there are enough resources in each cluster (taking into account its current *II*) to schedule all the instructions assigned to it. We have implemented this heuristic following the steps described in [3]. The second heuristic tries to produce a partition that can be scheduled, producing lower ED2 results. For that purpose, we will generate other partitions by moving nodes between clusters. These partitions are compared and the one likely to produce the lowest ED2 is selected. Hence, we need a way to predict which partitions will result in a schedule with the lowest ED2.

To obtain an estimate, a *pseudo-schedule* is used [3]. A pseudo-schedule is an approximate schedule which can be computed quite fast. It attempts to capture all of the characteristics of the final schedule. Thus, every time a new partition is generated, its pseudo-schedule is computed. We use it to estimate execution time. To estimate energy consumption, we use the model described in section 3.1.

With the pseudo-schedule and the static power model, we are able to compare two partitions: the one resulting in the lowest ED2 metric is selected. In case of a tie, we use the criteria described in [3].

5. Experimental Evaluation

In this section we will describe the experimental environment used to evaluate our scheme and present simulation results obtained. The proposed modulo scheduling technique has been implemented in the Open Research Compiler (ORC) [20]. We used the maximum optimization level, that is –O3.

For the purpose of this study, more than 4000 loops taken from SPECfp2000 benchmarks have been used.

In particular, we limit our focus to fortran applications because of the difficulties encountered by ORC to properly disambiguate references in C programs where pointers are frequently used. These references turn the whole *DDG* of a loop into one big recurrence, and thus, MS is ineffective. In particular, the IPC obtained for these programs for a non-clustered microarchitecture with an issue width of 12 instructions per cycle is lower than 1. In those cases, the use of acyclic scheduling techniques becomes more appropriate. The loops used for this evaluation are those considered for sofware-pipelined by the ORC.

We will report results for a microarchitecture with 4 floating point FU's, 4 integer FU's, 4 memory ports and 64 registers. These resources will be equally split into 4 clusters (i.e., 1 floating point FU, 1 integer FU, 1 memory port and 16 registers per cluster). Hence, all of the clusters will have the same design. For the intercluster connection network, we assume a 1-cycle latency register bus. We report results for 1 and 2 buses. The memory hierarchy is shared by all clusters and all cache accesses are considered hits. The latency assumed for the instructions, and their average energy consumption relative to an integer add, is reported. Since we are interested in simultaneously increasing performance, while at the same time reducing energy consumption, we will use ED2 to compare the different designs.

As described in section 3, we will use a reference homogeneous clustered microarchitecture to compute the power consumption. We assume a frequency of 1GHz, a supply voltage of 1V and a threshold voltage of 0.25V. For this baseline microarchitecture, one third of the energy is consumed by the memory hierarchy and 10% by the intercluster connection network. Leakage accounts for one third of the energy consumed by the clusters, two thirds for the cache and 10% for the intercluster connection network (because the bus usage is very high).

Concerning heterogeneous microarchitectures, we will assume one fast cluster and three slow clusters. For the fast cluster cycle times, we allow 0.9; 0.95; 1; 1.05; 1.1 times the cycle time of the reference homogeneous microarchitecture. For the slower clusters, we allow a cycle time of 1; 1.25; 1.33; 1.5 times the cycle time of the fast clusters. The cache frequency is set equal to the fastest cluster because memory latency is already a critical issue in processor design. Delaying all memory instructions would have a significant impact on performance. In particular, the latency of recurrences will increase, which in turn, would impact the *MIT*. Similarly, the inter-connection network frequency is also set equal to the frequency of the fastest cluster because, for modulo scheduled loops, the buses are commonly a constrained resource. Regarding the supply voltages, they can be scaled in the range of 0.7V to 1.2V for the clusters, 0.8V to 1.1V for the intercluster connection network, and 1V to 1.4V for the cache. The supply voltages are higher for the cache because its static energy consumption is large.

5.1 Optimum Homogeneous Microarchitecture

Before testing different heterogeneous configurations, we need to find the homogeneous configuration that minimizes ED2. Otherwise we could not be sure whether the benefits of the proposed technique were due to heterogeneity or they could also be achieved by using a homogeneous configuration with a different voltage and frequency.

To select the optimum homogeneous microarchitecture we have chosen to use the model described in section 3. When testing different homogeneous designs, this model is even more accurate because we can assume that the scheduling of a loop will be the same for any homogeneous design. Hence, the energy consumed by different homogeneous designs can be precisely estimated with the formulae presented in section 3.1. Similarly, the number of cycles used is the same for any homogeneous design and the difference in execution time comes from the differences in cycle time.

For the two different configurations studied (with 1 and 2 buses), the baseline design chosen will be the optimum homogeneous design for that configuration.

5.2 Heterogeneous Microarchitectures

Figure 6 shows the ED2 for different heterogeneous configurations, each one normalized to the ED2 for an optimum homogeneous microarchitecture. The main conclusion is that heterogeneity reduces ED2 for all the benchmarks and for the tested microarchitectures. In particular, the benefits are 15% on average. For some programs such as 200.sixtrack, ED2 is reduced by more than 35%; for 187.facerec the benefit is around 30% and for 189.lucas it is 20-25%. This is due to the fact that in these programs, most of the execution time is spent in loops that are recurrence constrained. Heterogeneity can be effectively exploited when considering recurrence constrained loops. Within these loops, there is a subset of instructions critical for performance: the instructions that are in the recurrences that have the longest latency. Usually, these instructions are a small part of the total number of instructions. Our algorithm that selects frequencies and voltages detects that the loops are recurrence constrained and selects a configuration such that there is a large difference in the frequency of the fastest cluster and the remaining slower clusters. Then, the proposed modulo scheduling scheme is able to place instructions contained in critical recurrences in the

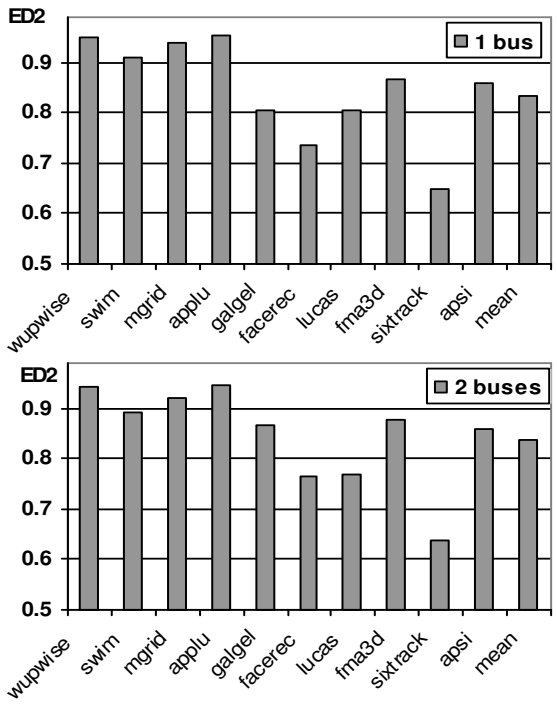


Figure 6: ED2 of the heterogeneous approach normalized to the optimum homogeneous.

faster clusters, whereas the remaining instructions are placed in the slower clusters (where they will consume significantly less energy). Hence, the execution time can be reduced and, at the same time, the energy consumption diminishes.

Table 2 presents the characteristics of the loops scheduled for the homogeneous microarchitecture when we use a single bus. We classify execution time based on the percentage of time that each program runs on resource-constrained loops (*resMII*>*recMII*), recurrence-constrained loops (*resMII*<*recMII*) and loops where it is not clear whether there will be resource or recurrence constraints. The latter group is composed of loops where the *recMII* is slightly larger than the *resMII*. These loops are, in principle, recurrence constrained. However, if an heterogeneous configuration is used, then there will be fewer resources available (because in the slower clusters, fewer instructions can be scheduled for the same *IT*) and hence they can easily become resource constrained. As we can see, the three programs that achieved the most benefit spent a large part of their execution time executing recurrence-constrained loops.

There are two programs (191.fma3d and 301.apsi), for which the benefits obtained are not as large (around 15%), spending most of their execution time executing recurrence-constrained loops. For these two programs, the speed-up achieved is close to the speed-up achieved for the three programs for which the largest benefits are obtained. However, the amount of energy saved is not that significant. This is due to the fact that in the critical recurrences in the loops of these two programs there are more instructions than in the recurrences of the former three. Hence, to obtain a speed-up, more instructions have to be placed in the faster clusters. These instructions will then consume more power, and the energy savings will be reduced.

The smallest benefits are obtained for 168.wupwise and 173.applu (5%). In the later case, the loops in 173.applu that have the largest impact on execution time are executed a small number of times. That makes *it_length* as important a factor as the *IT* for reducing execution time. Heterogeneity does not effectively reduce *it_length* because a significant number of the instructions do not have enough slack to be placed in the slower clusters without impacting overall execution time. Inspecting Table 2, for 168.wupwise we can see that the majority of its execution time is spent in loops that have a similar *resMII* and *recMII*. When selecting the voltage to use, if we determined that a heterogeneous configuration should be used, these loops would then become resource constrained and the *MIT* would increase. Therefore, our algorithm will choose a configuration where all the clusters run at the same frequency. The benefits come from adjusting the frequency and the voltages of the different components in order to minimize the energy consumption.

Finally, there are two programs whose loops are resource constrained, namely 171.swim and 172.mgrid. For resource-constrained loops, heterogeneity cannot benefit performance nearly as much as in the case of recurrence-constrained loops. This is because in the resource-constrained loop, all the instructions have a similar impact on execution time. Hence, it is not possible to reduce execution time by placing only a small number of instructions in faster clusters.

Nevertheless, we studied a case for which heterogeneity could be beneficial for resource-constrained loops. Assume a loop where the most constrained resources are integer FU's. Assume there are 21 instructions that make use of integer FU's. For a homogeneous microarchitecture with 4 integer FU's, the *MII* would be 6. However, for a heterogeneous microarchitecture, we can choose a configuration such that the *MII* for one faster cluster is 6 and the *MII* for the remaining three slower clusters is 5. Hence, we could use a lower frequency for 3 clusters and reduce energy consumption. However, it is often difficult to take advantage of this feature. First, the number of instructions that determines the most constrained resource varies from loop to loop and it is difficult to find a heterogeneous design that adapts well to all the loops in a program. Moreover, resource-constrained loops also suffer from register pressure because additional ILP is found. For register-constrained loops,

	recMII < resMII	resMII≤ recMII && recMII< 1.3 resMII	1.3resMII ≤ recMII
168.wupwise	14.04%	68.76%	17.2%
171.swim	100%	0%	0%
172.mgrid	95.54%	0%	4.46%
173.applu	31.94%	6.17%	61.89%
178.galgel	33.27%	9.18%	57.55%
187.facerec	16.59%	0%	83.41%
189.lucas	32.13%	0.02%	67.85%
191.fma3d	15.22%	2.96%	81.82%
200.sixtrack	0.08%	0%	99.92%
301.apsi	15.50%	3.37%	81.13%

Table 2: percentage of execution time spent in loops with resource/recurrence constraint loops.

we have observed that the best solution is to distribute 0register pressure equally among clusters and, thus, the best configuration is usually the one where all clusters run at the same frequency. The algorithm that selects voltages and frequencies can detect when programs are register constrained and so it chooses the same frequency for all clusters. The benefits achieved for these two programs (close to 10%) are due to energy savings. Since the IPC of resource constrained loops is higher than for recurrence constrained loops (because more ILP can be exploited), dynamic energy accounts for a larger percentage of the total energy consumption. Hence, the algorithm selects a lower frequency. Execution time is slightly longer, but significant energy savings are reaped. In particular, for both programs, execution time is increased by around 5%, but the energy consumption is reduced around 15%.

It is also interesting to note that the benefits obtained for all programs are similar, independent of whether 1 or 2 buses are used. In fact, the benefit depends heavily on the characterisitcs of the workload. Other configurations have also been tested and we have found similar results.

5.3 Sensitivity Analysis

In this section we study the impact of varying some of the parameters in the baseline. In Figure 7 we can see the ED2 when a different number of frequencies are supported in each component of the microarchitecture. We consider an infinite number of frequencies, 16, 8 and 4 frequencies. As we can see, allowing for 16 different frequencies in each component of the microarchitecture provides the same benefits as allowing for any frequency (the differences are under 0.1%). Besides, if only 8 frequencies are allowed, the ED2 degradation is smaller than 1%. For 4 frequencies the degradation grows to 2%.

If a microarchitecture provides only a small number of frequencies, different techniques could be applied to

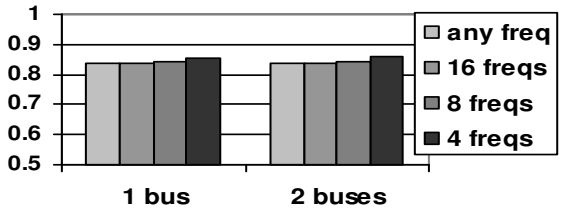


Figure 7: ED2 for different number of frequencies supported.

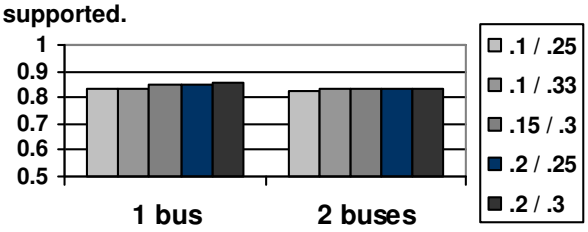


Figure 8: ED2 varying the percentage of energy consumption of the ICN and the cache.

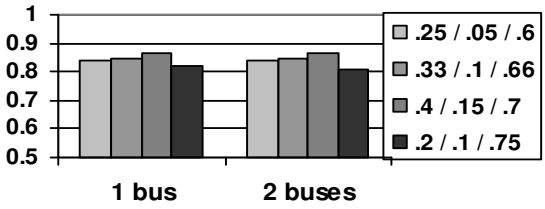


Figure 9: ED2 for different leakage assumptions for the baseline architecture (cluster/ICN/cache).

offset this negative impact. First, the selection of the frequencies supported should be done more carefully. In this work we only support frequencies that allow for synchronization with the slowest *IT*'s without taking into account whether these frequencies are used often or not. A study of which frequencies appear most often could be done. In addition, the scheduling approach could support features to reduce the impact of increasing the *IT* due to synchronization. For this purpose, loop unrolling is efficient. The *MIT* of an unrolled loop is multiplied and the penalty associated with increasing the *IT* due to synchronization is reduced. Besides, the unroll factor can be chosen in such a way that the resulting *IT* allows for synchronization. This proves that the additional hardware required to support frequency scaling could be assumed.

In Figure 8 we present ED2 for a system where we change the energy consumption percentages for the cache and the intercluster connection network of the reference homogeneous microarchitecture. The columns are labeled with two numbers that represent the portion of the total energy consumption due to the interconnection network and due to the cache, respectively. Thus, ".1 / .25" stands for the configurations obtained from a reference homogeneous microarchitecture where 10% of the total energy consumption is due to the interconnection network and

25% is due to the cache. Note that for each different reference homogeneous microarchitecture a different optimum homogeneous is computed and used for comparison. As can be seen in the figure, results vary slightly.

In Figure 9 we present ED2, varying the amount of energy consumption due to leakage (as a percentage of the total energy consumed) across each of the components of the baseline architecture. The columns are labeled with three numbers, representing the portion of the total energy due to leakage in the clusters, the ICN and the cache, respectively. Changing these percentages has little impact. From inspecting all 3 figures, we see that our scheme is somewhat independent of the assumptions made for the baseline microarchitecture.

6. Related Work

There is limited prior work related to heterogeneous clustered designs. Baniasadi and Moshovos[4] proposed a heterogeneous clustered processor. However, the microarchitecture neither employs dynamic voltage (*DVS*) nor frequency scaling (*DFS*). In addition, they target a dynamically-scheduled system. Hence, they do not deal with instruction scheduling techniques but instead propose a scheme to dynamically distribute instructions. In contrast, in this paper we present a highly flexible statically-scheduled heterogeneous design where clusters can be reconfigured to adapt to the characteristics of each application. Moreover, a set of compile-time techniques is proposed to distribute instructions to achieve high-performance while reducing power consumption.

The most closely related work to our proposal includes the work of Muralimanohar et al.[22] where they describe a heterogenous clustered processor that is able to perform DVS and DFS. This work also target dynamically-scheduled. Their work focuses on mechanisms to dynamically decide the most appropriate frequency/voltage configuration for each cluster. They do not propose novel approaches to perform distribution of the instructions among clusters. Instead, our poprosal targets both problems by: (1) proposing a novel method to perform off-line reconfiguration of the microarchitecture at the program level, and (2) proposing a new scheme for instruction distribution to produce improvements in ED2.

Off-line frequency/voltage reconfiguration has been studied for MCD processors [22], where the goal was to identify non-critical sections of code working at a coarse granularity, e.g. function level. Then, configuration selection is performed for these non-critical sections by running them into a power-oriented configuration. Hence, the processor may achieve high-performance while minimizing energy consumption. In contrast, our methodology performs off-line reconfiguration of the different domains based on the characteristics of software-pipelined loops. Moreover, careful instruction distribution is performed to select non-critical instructions at a fine-grain granulatity.

On the other hand, several modulo scheduling approaches targeting clustered VLIW architectures have been proposed [24][11][29][7][2][3]. All these works have targeted homogeneous microarchitectures.

To the best of our knowledge, this paper is the first proposal for an heterogenous statically-scheduled clustered processor. Modulo scheduling combined together with techniques to exploit heterogeneity by using voltage and frequency scaling has not been studied to date.

7. Conclusions

In this work we present a rigorous study of heterogeneous clustered VLIW architectures. We present a qualitative and quantitative study that clearly illustrates the benefits of using heterogeneity to further reduce execution time and energy consumption, significantly decreasing ED2.

In order to effectively exploit the capcabilities of the a heterogeneous clustered microarchitecture, we have developed a number of compiler algorithms. First, we propose a technique for selecting the most appropiate frequencies and voltages for all the components of the system. This technique is based on a fast and accurate model for estimating the energy consumption and the execution time of an application . Second, we describe a MS technique that effectively utilizes of the potential benefits of a heterogeneous system by placing critical instructions in fast clusters to achieve high performance and the rest of the instructions in low-power clusters to reduce energy consumption.

Our results show significant ED2 improvements. In particular, the average benefit in ED2 is over 15%, while much greater benefits are achieved for selected programs (e.g., 35% for 200.sixtrack).

8. References

[1] A. Agarwal, M.S. Hrishikesh, S.W. Keckler and D. Burger, “Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures”, in *Proc. of the 27th Int. Symp. on Computer Architecture*, June 2000.

[2] A. Aletà, J.M. Codina, J. Sánchez and A. González. “Graph-Partitioning Based Instruction Scheduling for Clustered Processors”, in *Proc. of 34th Int. Symp. On Microarchitecture*, Dec 2001.

[3] A. Aletà, J.M. Codina, J. Sánchez, A. González and D. Kaeli. “Exploiting Pseudo-schedules to Guide Data Dependence Graph Partioning”, in *Proc. of the Int. Conf. on*

Parallel Architectures and Compilation Techniques (PACT'02), Sept 2002.
[4] A. Baniasadi and A. Moshovos, "Asymmetric-frequency clustering: a power-aware back-end for high-performance processors", in *Proc. of the Int. Sym. on Low Power Electronics and Design, 2002.*
[5] A. Capitanio, D. Dytt and A. Nicolau, "Partitioned Register Files for VLIWs: A Preliminary Analysis of Tradeoffs", in *Proc. of 25th. Int. Symp. on Microarchitecture*, pp. 192-300, 1992.
[6] A. Charlesworth, "An Approach to Scientific Array Processing: the Architectural Design of the AP120B/FPS-164 Family", *Computer*, 14(9):18-27, 1981.
[7] J.M. Codina, J. Sánchez and A. González. "A Unified Modulo Scheduling and Register Allocation Technique for Clustered Processors", in *Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'01)*, Sept 2001.
[8] R. Ellis, "Bulldog: A Compiler for VLIW Architectures", *MIT Press*, pp. 180-184, 1986.
[9] K. Farkas, "Memory-System Design Considerations for Dynamically Scheduled Microprocessors", PhD thesis, *University of Toronto*, 1997.
[10] P. Faraboschi, G. Brown, J. Fisher, G. Desoli and F. Homewood, "Lx: A Technology Platform for Customizable VLIW Embedded Processing", *in Proc. of the 27th Int. Symp. on Computer Architecture*, pp. 203-213, June 2000.
[11] M.M. Fernandes, J. Llosa and N. Topham, "Distributed Modulo Scheduling", in *Proc. of Int. Symp. on High-Performance Computer Architecture*, pp. 130-134, Jan. 1999.
[12] T. Fischer, F. Anderson, B. Patella, S. Naffziger, "A 90nm Variable-Frequency Clock System for a Power-Managed Itanium-Family Processor", in *Proc. of* ISSCC, March 2005.
[13] J. Fridman and Z. Greenfield, "The TigerSharc DSP Architecture", *IEEE Micro*, pp. 66-76, Jan-Feb. 2000.
[14] P.N. Glaskowsky, "MAP1000 unfolds at Equator"", Microprocessor Report, 12(16), Dec. 1998.
[15] R. Ho, K. Mai and M. Horowitz, "The Future of Wires", in *Proc. of the IEEE*, April 2001.
[16] S. Jang, S. Carr, P. Sweany and D. Kuras, "A Code Generation Framework for VLIW Architectures with Partitioned Register Banks", in *Proc. of 3rd. Int. Conf. on Massively Parallel Computing Systems*, 1998.
[17] G. Karpis and V. Kumar, "Analysis of Multilevel Graph Partitioning", in *Proc. of 7th Supercomputing Conference*, 1995.
[18] V. Kathail, M. Schlansker and B. Rau, "HPL-PD Architecture Specification: Version 1.1". Technical Report HPL-93-80, Hewlett-Packard Laboratories, February 2000.
[19] W. Lee, D. Puppin, S. Swenson, S. Amarasinghe, "Convergent Scheduling", *in Proc. of 35th Int. Symp. on Microarchitecture*, December 2002.
[20] C. Lim, R. Ju, J. Zhang, L. Chen, X. Feng, C. Wu, "Open Research Compiler (ORC): A Compiler Infrastructure for Research". *Presented at the 17th International Workshop on Languages and Compilers for Parallel Computing, LCPC04*, September 2004.
[21] J. Llosa, E. Ayguadé, A. González and M. Valero. "Swing Modulo Scheduling", in *Proc. of Int. Conf. on Parallel Architectures and Comillation Techniques (PACT'96)*, Oct 1996.
[22] G. Magklis, M.L. Scott, G. Semeraro, D.H. Albonesi, and S. Dropsho, "Profile-based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor", *in Proc of 30th Int. Symp. on Computer Architecure* (ISCA).
[23] N. Muralimanohar, K. Ramani, and R. Balasubramonian, "Power Efficient Resource Scaling in Partitioned Architectures through Dynamic Heterogeneity", in *Proc. Of IEEE Int. Symp on Performance Analysis of Systems and Software (ISPASS)*, 2006.
[24] E. Nystrom and A. E. Eichenberger, "Effective Cluster Assignement for Modulo Scheduling", *in Proc. of the 31st Int. Symp. on Microarchitecture*, pp. 103-114, 1998.
[25] T. Olsson, P. Nilsson, T. Meincke, A. Hemani and M. Torkelson, "A Digitally Controlled Low-Power Clock Multiplier for Globally Asynchronous Locally Synchronous Designs", *in Proc.* of ISCAS'2000.
[26] G.G. Pechanek, and S. Vassiliadis, "The ManArray Embedded Processor Architecture," *in Proc. of the 26th. Euromicro,* Vol. I, pp.348-355, Sept. 2000.
[27] B.R. Rau and C. Glaeser, "Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Schientific Computing", *in Proc. of 14th Microprogramming Workshop*, pp. 183-197, October 1981.
[28] B.R. Rau, "Iterative Modulo Scheduling", *Hewlett-Packard Company*, 1995
[29] J. Sánchez and A. González, "The Effectiveness of Loop Unrolling for Modulo Scheduling in Clustered VLIW Architectures", *in Proc. of the 29th Int. Conf. on Parallel Processing*, Aug. 2000.
[30] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling", in *Proc of the 8th Int. Symp. on High-Performance Computer Architecture*, Feb. 2002
[31] Texas Instruments Inc., "TMS320C62x/67x CPU and Instruction Set Reference Guide", 1998.
[32] H. Zhong, K. Fran, S. Mahlke and M. Schlansker, "A Distributed Control Path Architecture for VLIW Processors", in *Proc. of Int. Conf. on Parallel Architectures and Compillation Techniques,* 2005.