Cluster Generation and Scheduling for Instruction (L0) Clusters[*]

M. Jayapala, F. Barat, T. Vander Aa, G. Deconinck, F. Catthoor and H. Corporaal
{mjayapal,fbaratqu,tvandera,gdec}@esat.kuleuven.ac.be,
catthoor@imec.be, h.corporaal@tue.nl
ESAT/ELECTA, Kasteelpark Arenberg 10, K.U.Leuven,
Leuven-Heverlee, Belgium - 3001

30 August 2003

Abstract

Clustered L0 buffers are an interesting alternative to reduce energy consumption in the instruction memory hierarchy of embedded VLIW processors. Currently, the synthesis of L0 clusters is performed as an hardware optimization, where the compiler generates a schedule and based on the given schedule L0 clusters are generated. Since, the result of the clustering depends on the given schedule, it offers an interesting design space to explore the effects of clustering by altering the schedule to increase energy efficiency. This paper presents a study indicating the potentials offered by scheduling for L0 clusters in terms of L0 buffer energy reduction. The list scheduler is extended to recognize the L0 clusters, and based on a few simple heuristics the operations are assigned to certain L0 clusters, followed by an iterative methodology to reduce L0 buffer energy consumption. The simulation results indicate that potentially up to 10% of the L0 buffer energy can be reduced by scheduling for L0 clusters with a simple heuristic.

1 Introduction

Current embedded systems for multimedia applications like mobile and hand-held devices, are typically battery operated. Therefore, low energy is one of the key design goals of such systems. Many such systems often rely on VLIW ASIPs. However, power analysis of such processors indicate that a significant amount of power is consumed in the instruction caches. L0 buffering is an effective scheme to reduce energy consumption in the instruction memory hierarchy [1]. For multimedia applications storing the loop intensive parts of the code in a small L0 buffer instead of the big instruction cache, energy can be reduced significantly. While this reduction is substantial, further optimizations are still necessary to ensure such high energy efficiency in the future processors. With orthogonal optimizations applied on different aspects of the processor like the datapath, register files and data memory hierarchy, the overall processor energy reduces. However the instruction cache energy, including the L0 buffer, is bound to increase. Of the two main contributors of energy consumption in the instruction memory hierarchy, the L0 buffers are the main bottleneck. To alleviate the L0 buffer energy bottleneck, a clustered L0 buffer organization has been proposed in the recent past [2]. Essentially, in a clustered L0 buffer organization, the L0 buffers are partitioned and grouped with certain functional units in the datapath to form an instruction cluster or an L0 cluster [2].

Motivation to Schedule for L0 Clusters Currently, the generation of L0 clusters is a hardware optimization. For a given application and a corresponding schedule, the clustering tool analyzes the functional unit activation trace and generates clusters that optimizes energy [2]. The left-side in Figure 1, illustrates the flow for generating clusters. For a given application, a pre-compiler tool analyzes the profile and identifies certain interesting parts[1] of the code to be mapped on to the L0 buffers. In the example, a certain for-loop is mapped on to the L0 buffer. The compiler then generates a certain schedule for the selected parts of the code (for-loop in the example). Here, the schedule is generated for an 8-issue VLIW processor, which has 3 instructions with 4 operations in first instruction and 1 operation in second and third instructions. The clustering tool generates L0 clusters that minimizes the L0 buffer energy, and which is the optimum for this given schedule. No other clustering will have a lower energy consumption for this schedule. In the example, the above schedule generates 4 clusters with energy consumption of 1.0 units. The associated functional unit grouping and the L0 buffer partition sizes are as indicated in the Figure 1. For example, functional units (or issue slots) 3, 5 and 7 are grouped into a single cluster and it has an L0 buffer partition of width 3 operations and a depth of one word.

However, the resulting clusters from the clustering tool are sensitive to the schedule. For instance in the above example, if the schedule is slightly modified as indicated in Figure 1, then the same clustering tool would now produce different clusters: 3 clusters, with different functional unit grouping and L0 buffer partitions. The L0 buffer energy is also reduced from 1.0 units to 0.91 units. This is a good motivation to investigate the potentials offered by scheduling to reduce L0 buffer energy.

2 Scheduling for L0 Clusters

In order evaluate the scheduling freedom and L0 cluster generation across instructions and extending up to basic blocks,

[*]This work is in part supported by MESA under MEDEA+ program

[1]Mainly loops, but a group of basic blocks within a loop can also be mapped.

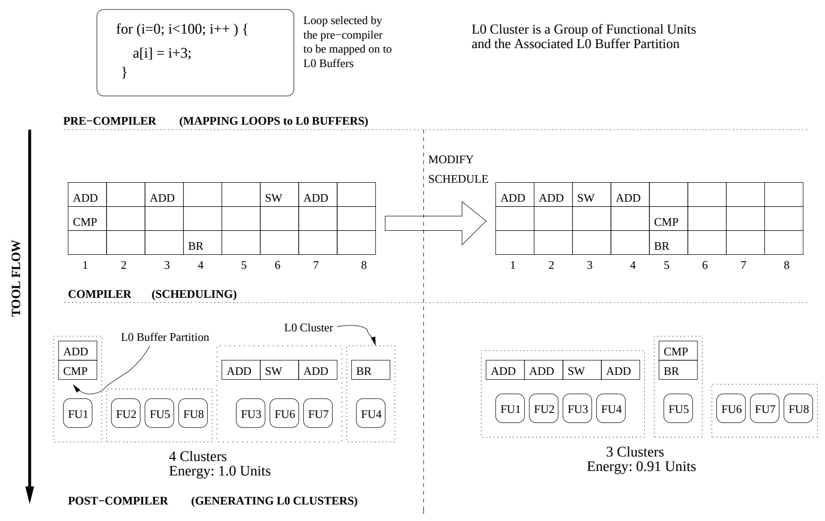
for (i=0; i<100; i++) {
a[i] = i+3;
}
Loop selected by the pre−compiler to be mapped on to L0 Buffers
L0 Cluster is a Group of Functional Units and the Associated L0 Buffer Partition
PRE−COMPILER (MAPPING LOOPS to L0 BUFFERS)
TOOL FLOW
MODIFY SCHEDULE
ADD
CMP
ADD
BR
SW
ADD
1 2 3 4 5 6 7 8
ADD ADD SW ADD
CMP
BR
1 2 3 4 5 6 7 8
COMPILER (SCHEDULING)
L0 Buffer Partition
L0 Cluster
FU1 FU2 FU5 FU8 FU3 FU6 FU7 FU4
4 Clusters
Energy: 1.0 Units
FU1 FU2 FU3 FU4 FU5 FU6 FU7 FU8
3 Clusters
Energy: 0.91 Units
POST−COMPILER (GENERATING L0 CLUSTERS)

Figure 1: Example Illustrating Scheduling Sensitivity on L0 Cluster Generation

the following iterative scheme as described in Figure 2 is employed. During the first iteration, the compiler generates a certain schedule for an unclustered machine (no L0 clusters). Based on the given schedule, a certain optimal L0 clustering is generated. The functional units and the L0 buffers as described in the machine description are regrouped according to the L0 clusters obtained in the first iteration. In the following iterations, the compiler assigns the operations to L0 clusters based on a certain heuristic, and thus alters the schedule. Now, for the new schedule the instruction clusters are obtained, and the most optimal clustering result is chosen for the next iteration. This process is repeated for a certain preset number of iterations.

Extended List Scheduler The core of algorithm of the list scheduler that is relevant to the operation assignment is described in the Algorithm 1. For a given basic block, the scheduler generates a list of operations based on certain priority. Now, each operation in that list has to be assigned to a certain functional unit. In the extended list scheduler, instead of choosing the first available functional unit, a list of possible L0 clusters is created (*L0_cluster_list*). This L0 cluster list is now prioritized in descending order based on a simple heuristic, as explained in the following section. A matching slot (*get_non_conflicting_slot*) is now chosen among the prioritized L0 cluster list. Once a matching slot is found, the current operation is bound to that functional unit and the rest of the scheduling process (*schedule_op*) takes over.

Heuristics for Prioritizing L0 Cluster List

Given a certain operation the problem is to choose a certain instruction cluster in such a way that the modified schedule may lead to a more energy efficient clustering solution. One of the following two heuristics can be employed to assign an operation to a cluster in an intelligent way.

The first heuristic is to assign operations to a cluster so that, if a cluster is used, all the following operations are tried to be assigned to the same cluster. This heuristic comes from the observation that, by continuing to use the same cluster, the remaining clusters can be made inactive, thus saving energy. The second heuristic is similar to the previous one. Except that, after the cluster list is prioritized, if an operation cannot be assigned to the highest priority cluster (due to some conflicts), the schedule time of the current operation is increased until the current operation is assigned to that cluster. In the previous heuristic, if the operation cannot be assigned to the highest priority cluster, then the lower priority clusters are tried, but the schedule time is not increased.

Algorithm 1 List Scheduler Extension for Assigning Operations to Instruction Clusters

```
Derive_Schedule (Basic Block){
    ListOp = GenerateList(Basic Block);
    foreach (Op in ListOp){
        L0_cluster_list = build_L0_cluster_list(Op);
        prioritize_L0_icluster_list(Op, PrevOp, L0_icluster_list);
        get_non_conflicting_slot(Op, L0_cluster_list);
        schedule_op (Op);
    }
}
```

3 Experimental Results

This section outlines the simulation results of scheduling for L0 clusters. Each benchmark is passed through the tool chain as indicated in Figure 2. The first phase is the pre-compiler, which maps the interesting parts of that application on to the L0 buffers. The second phase is the compiler suite, which is an extended version of the Trimaran framework. The list scheduler in the backend of this framework has been extended to implement the operation assignment phase as described in the previous section. The third phase is the post-compiler, wherein the tool generates an optimal L0 clustering for a given schedule. The number of iterations is preset to ten. The energy models of the L0 buffers are derived from Wattch for a 0.18um

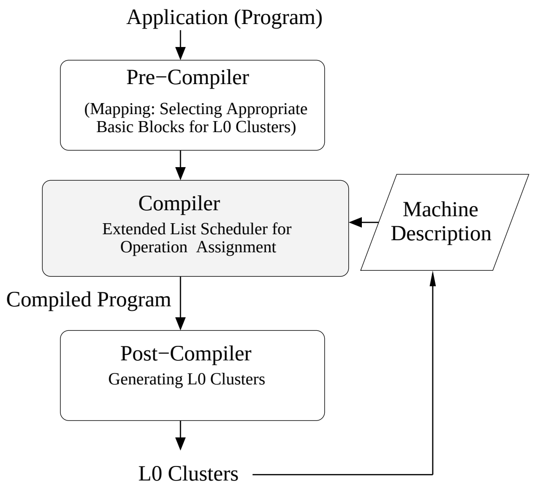

Figure 2: An Iterative Methodology for Operation Assignment (Scheduling) and Clustering

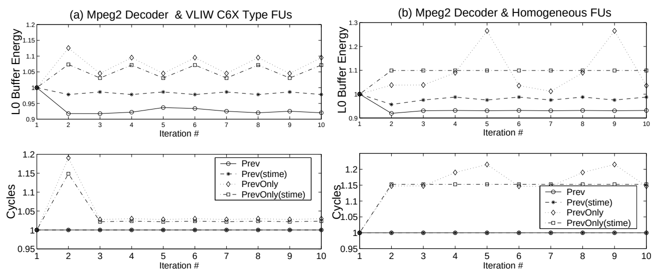

Figure 3: L0 Buffer Energy and Performance Figures for Mpeg2 Decoder

technology. The architecture under consideration is an unclustered (datapath) VLIW processor with 8 functional units. Two datapath variants have been considered: (a) a datapath with functional units similar to the TI C6X (b) a datapath with homogeneous functional units.

Figure 3 shows the energy and performance results for the two datapath variants for the Mpeg2 Decoder benchmark. For all the heuristics, the energy and performance figures in the first iteration corresponds to the clustering result as applied to a default schedule (without considering the L0 clusters).

The curves corresponding to legend ’Prev’ represents the energy and performance figures for the heuristic: assigning a high priority to an L0 cluster to which the previous operation was assigned to. The figures show that, up to 10% of L0 buffer energy can be reduced in both the datapath variants without any loss of performance. Soon after the first iteration, the energy consumption fairly stabilizes in the remaining iterations. After the first iteration, the scheduler generates a schedule for the L0 clusters, as obtained from the first iteration. This schedule achieves the 10% energy reduction. Since the schedule times are not modified, the performance figures do not change either (refer Figure 3).

The curves corresponding to legend ’PrevOnly’ represents the energy and performance figures for the heuristic: assigning an operation to the L0 cluster to which the previous operation was assigned to, even if schedule time needs to be modified. The figures show that over the subsequent iterations, both energy and performance deteriorates. Also, the deterioration is worse for the homogeneous variant (Figure 3(b)), than the VLIW C6X variant (Figure 3(a)). Since the chances of a subsequent operation to be capable of being executed in the previous operation’s cluster is very high in a homogeneous variant, the energy and performance figures are worse than the figures for the C6X variant.

For each of the above two heuristics, another variant was experimented. Instead of applying the prioritizing-heuristic for all the operations, they were applied only to operations, if those operations were to be scheduled in the same cycle. The corresponding energy and performance figures are represented by legends ’Prev(stime)’ and ’PrevOnly(stime)’. The aim was to see if a smaller perturbation in the schedule (for the ’Prev’ heuristic, the schedule changes significantly after the first iteration) lead to a better solution. However, the smaller perturbation led to a smaller energy reduction, about 5% (Figure3(a) and (b)), and no further reduction was achieved during the subsequent iterations.

4 Future Work

In summary, this paper presents a study indicating the potentials offered by scheduling for L0 clusters in terms of energy reduction. The list scheduler is extended to recognize the L0 clusters, and based on a few simple heuristics the operations are assigned to certain L0 clusters, followed by an iterative methodology to reduce L0 buffer energy consumption. The simulation results indicate that potentially up to 10% of the L0 buffer energy can be reduced by scheduling for L0 clusters with a simple heuristic.

The future work would involve investigating multi-pass heuristics, where a certain cost function is evaluated when a few operations are assigned and depending on the cost value the operations may be reassigned. Current scheduler for VLIW processors employ modulo scheduling in one form or the other. Extending the modulo scheduler for L0 clusters would be another direction for further study. Datapath clusters and L0 clusters can coexist. However, current schemes for generating datapath clusters and L0 clusters are mutually exclusive, and the resulting clusters might be in conflict. The synchronicity between datapath and L0 clusters needs to be investigated.

References

[1] R. S. Bajwa, M. Hiraki, H. Kojima, D. J. Gorny, K. Nitta, A. Shridhar, K. Seki, and K. Sasaki, “Instruction buffering to reduce power in processors for signal processing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, pp. 417–424, December 1997.

[2] M. Jayapala, F. Barat, T. VanderAa, F. Catthoor, G. Deconinck, and H. Corporaal, “Clustered l0 buffer organization for low energy embedded processors,” in *Proc of 1st Workshop on Application Specific Processors (WASP), held in conjunction with MICRO-35*, November 2002.