PARLGRAN: Parallelism granularity selection for scheduling task chains on dynamically reconfigurable architectures *

Sudarshan Banerjee Elaheh Bozorgzadeh Nikil Dutt
Center for Embedded Computer Systems
University of California, Irvine, CA, USA
{banerjee,eli,dutt}@ics.uci.edu

ABSTRACT

Partial dynamic reconfiguration, often called RTR (run-time reconfiguration) is a key feature in modern reconfigurable platforms. While partial RTR enables additional application performance, it imposes physical constraints necessitating simultaneous scheduling and placement while mapping application task graphs onto such architectures. In this paper we present PARLGRAN, an approach that maximizes performance of application *task chains* by selecting a suitable granularity of data-parallelism for individual *data parallel* tasks. Our approach focusses on reconfiguration delay overhead and placement-related issues (such as fragmentation) while selecting individual data-parallelism granularity as an integral part of simultaneous scheduling and placement. We demonstrate that our heuristic generates high-quality schedules on an extensive set of over a 1000 synthetic experiments by comparing the results with an approach that tries to statically maximize data-parallelism, i.e., does not consider the overheads and constraints associated with partial RTR. A detailed case-study on JPEG encoding additionally confirms that blindly maximizing data-parallelism can result in schedules even worse than that generated by a simple (but RTR-aware) approach oblivious to data-parallelism.

Keywords: Partial dynamic reconfiguration, data-parallelism, granularity selection, linear placement, scheduling

1. INTRODUCTION

Reconfigurable architectures are popular for applications with intensive computation such as image processing, since a limited amount of logic can be customized to set up deep pipelines, and/or exploit more coarse-grain parallelism, etc. Partial dynamic reconfiguration, or, run-time reconfiguration (RTR) allows additional customization during application execution, making it possible to obtain increased performance [11]. Our overall goal is to maximize performance of applications represented as precedence-constrained task DAGs (directed acyclic graphs) on *single-context* architectures with partial RTR (Xilinx Virtex-II is a commercial instance of such architectures). Some key issues in mapping applications onto such devices are the significant reconfiguration delay overhead, physical (placement) constraints, etc.

In this paper, we focus on precedence-constrained *task chains*, common in image-processing applications [7], [4]. In such applications, area-execution time characteristics of key tasks such as IDCT, Quantize, etc, are predictable because of complete pipelining. Additionally, key tasks such as DCT are completely *data-parallel*, i.e., results of task execution on a block of data are completely independent of results when the same task is executed on a disjoint block of data. On an architecture with partial RTR, it is possible to improve application execution time by *dynamically* adjusting the parallelism granularity of such tasks, i.e., reconfiguring the architecture to instantiate multiple copies of such tasks *during application execution* – each copy (instance) uses an identical amount of HW resources, but processes only part of the data. Due to complete pipelining, execution time of such tasks is directly proportional to the volume of data processed, and thus, reducing the data volume proportionately improves (reduces) the application execution time. Note that on architectures with no partial RTR, the scope of exploiting such data-parallelism is much more limited – partial RTR enables resource reuse, significantly expanding the potential of exploiting data-parallelism.

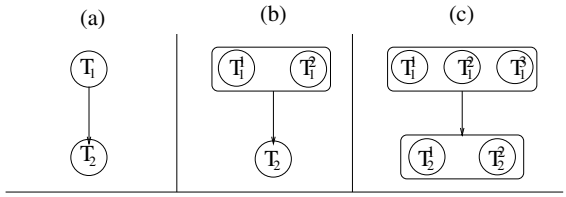


Figure 1: Granularity of individual data-parallel tasks

As an example, we consider a simple chain with two tasks, as shown in Figure 1. Assuming that there are enough resources to simultaneously execute 3 copies of task T_1 *or* 2 copies of task T_2, (b) and (c) show some possible task graph configurations after such a transformation. However, such a transformation can be quite costly on architectures with partial RTR– each new task instance (copy) adds a significant reconfiguration overhead. Therefore the transformations need to be guided by selecting the right granularity of parallelism that masks the reconfiguration overhead and maximizes performance. One important issue is that because of the reconfiguration penalty, multiple copies of a task may not be able to start at the same time– therefore, individual execution time (workloads) of the multiple copies may vary.

We propose an approach, PARLGRAN, that attempts to maximize application performance on architectures with partial RTR by choosing the right parallelism granularity for each individual data-parallel task. By granularity we mean both the **number of instances** (copies) of that task, and, the **workload** (execution time) of each copy. Our approach considers physical (placement) constraints, and utilizes configuration prefetch [10] to reduce the latency. The key constraints of such architectures necessitate joint scheduling and placement [8], [1]. Our approach therefore, incorporates granularity selection as an integral part of simultaneous scheduling and placement. To the best of our knowledge, ours is the first effort to solve this problem.

We validate the quality of our proposed approach against an approach that tries to statically maximize performance gain from data parallelism without considering the constraints and overheads due to partial RTR. A large set of over a thousand synthetic experiments demonstrates that the average improvement in schedule length by using our approach is over 15%. A detailed case study of JPEG encoding additionally confirms that the static parallelization approach

*This work was partially supported by NSF Grants CCR-0203813 and CCR-0205712

0-7803-9451-8/06/$20.00 ©2006 IEEE.

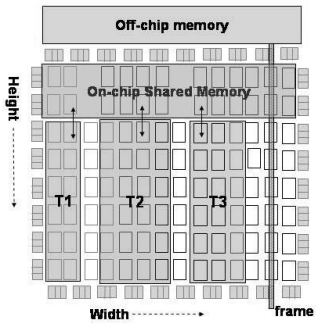


Figure 2: Target dynamic architecture

can end up generating schedules much worse than a simple (but RTR-aware) approach oblivious to data-parallelism.

2. RELATED WORK

There exists a large body of work for mapping task chains typical in image processing to reconfigurable architectures. A significant amount of the work such as [7] does not consider partial run-time reconfiguration (RTR). Recent work that does consider partial RTR such as [4], often focuses on multicontext architectures [9], where the reconfiguration overhead is negligible- unfortunately there is a very significant area overhead in such architectures. Other recent work such as [2] focusses on the key problem of task reuse as another technique for reducing the reconfiguration overhead, an aspect we do not address in this work. Our work focuses on reducing reconfiguration overhead by considering configuration prefetch as an integral aspect of joint scheduling and placement.

While there is a growing body of work in joint scheduling and placement on such architectures [3], [8], they typically ignore key architectural constraints such as the resource contention due to a single reconfiguration controller, prefetch to reduce the latency, etc. Ignoring these key issues makes the problem closer to the rectangle packing problem [13] and does not realistically exploit RTR.

Of course, there is a vast body of knowledge in the compiler domain about extracting parallelism from programs at different levels of granularity [14]. However, detailed consideration of placement and other architectural aspects related to partial RTR make our work significantly different.

3. PROBLEM OVERVIEW

Target architecture: Our target dynamically reconfigurable device as shown in Figure 2 consists of a set of configurable logic blocks (CLB) arranged in a two-dimensional matrix. The basic unit of configuration for such a device is a frame spanning the height of the device. A column of resources consists of multiple frames. A task occupies a contiguous set of columns. The reconfiguration time of a task is directly proportional to the number of columns (frames) occupied by the task implementation. One key constraint is that only one task reconfiguration can be active at any time instant. An example of our target device is the Xilinx Virtex-II series where constraints such as dynamic tasks occupying a contiguous set of columns are critical for realization of partial run-time reconfiguration.

Application specification: A task T_i executing on such a system can be represented as a 3-tuple (c_i, t_i, r_i) where c_i is the number of resource columns occupied by the task, t_i and r_i are the execution time and reconfiguration overhead respectively. Each task needs to be reconfigured before its execution is scheduled. The physical constraints on such a device necessitates joint scheduling and placement [8], [1].

In image processing applications, we often find chains (linear sequences) of such tasks. For a chain of n tasks, $(T_1..T_n)$, each task in the chain has exactly one predecessor and one successor. Of course, the first task, T_1, has no predecessor, and the last task, T_n, has no successor. A predecessor task utilizes a shared memory mechanism to communicate necessary data to its successor– this shared memory can be physically mapped to local on-chip memory and/or off-chip memory depending upon memory requirements of the application.

Our overall goal is to maximize performance (minimize schedule length) under physical and architectural constraints, given a resource constraint of C columns available for the application, $C < \sum_{i=1}^{n}(c_i)$. An additional goal is that our approach should have a low computational overhead.

4. MOTIVATION

Ideally, the degree of parallelism for a data-parallel task is limited only by the availability of HW resources. Let us consider a chain with only a single task T_1 that executes in time t_1 using c_1 columns. Given a resource constraint of C columns, performance is maximized when this task is instantiated $\lfloor C/c_1 \rfloor$ times, as shown in Figure 3. In this figure, the X-axis represents the columnar area constraint, C, and, the Y-axis represents the schedule length. For sequential tasks (0 degree of data-parallelism), the execution of task T_i is represented as E_i and the reconfiguration of task T_i is represented as R_i, as in Figure 3 (a). For data-parallel tasks, we additionally denote the execution of j-th instance (copy) of the task as E_i^j and the reconfiguration for this instance (copy) as R_i^j, as shown in Figure 3 (b). For uniform treatment, we assume that the compilation cost includes reconfiguration overhead for the first task in the chain, T_1– the schedule length is always computed from beginning of execution of T_1. However, this *ideal* performance gain is typically not achievable while considering realistic issues on such architectures, as discussed next.

Reconfiguration overhead: For modern *single-context* architectures that support partial RTR, the large reconfiguration delay is a key bottleneck in achieving *ideal* parallelism. To illustrate this, we consider Figure 4. We assume that the reconfiguration controller is available at the beginning of the execution of the first copy of the task. Next we instantiate two copies of this task with the intention of equally distributing the workload (execution time). However, execution of the second copy E_1^2 can start only after the reconfiguration overhead, r_1. Thus, instead of the ideal workload of $t_1/2$, the workload of the second task is only: $(t_1 - r_1)/2$, leading to less performance improvement than expected.

For a single task, a simple equation suffices to compute the best performance improvement, leading to the following lemma:

LEMMA 1. *For parallelizing a task into j instances, and given that the reconfiguration controller is available at the beginning of execution of the first instance, the best performance (least execution time) is obtained when the workload (execution time) of the j-th instance is:* $((t_1 - r_1 * (j * (j-1)/2))/j$.

Due to lack of space, detailed proof is in [16]. However, it is important to note the following key ideas, evident from Figure 4.

- Even if sufficient HW resources are available, the large reconfiguration delay may prevent further performance improvement if more than a few copies of a task are instantiated. In Figure 4, the 4'th copy does not improve performance (shorten schedule length).
- Maximizing performance involves *unequal* workload (execution time) distribution between multiple copies of a task, to compensate for the reconfiguration overhead.

Next, we consider the additional complications introduced by precedence constraints.

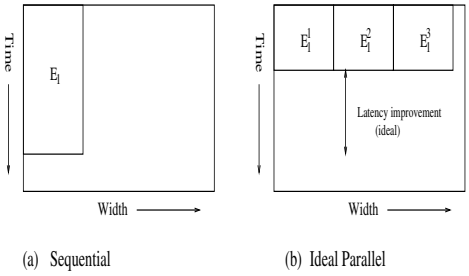


Figure 3: "Ideal" parallelism

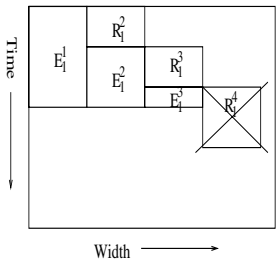


Figure 4: Parallelism degree determined by overhead

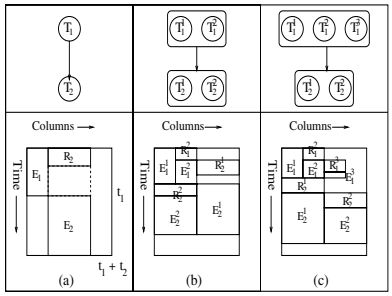


Figure 5: Precedence constraints- choices

Precedence constraints: For precedence-constrained tasks, simple equations such as Lemma 1 do not work any longer – we now need to consider the interaction of the resource demands of the tasks, as shown in Figure 5 for a simple chain with two tasks T_1 and T_2. The HW resource constraint allows three copies (instances) of T_1, **or**, two copies of T_2 to be executing simultaneously. We show some of the possible schedules and their transformed task graphs in Figure 5 (b) and (c). Note that in our execution model, *all copies of a parent task must finish execution before any copy of a child task starts execution*. Also, Figure 5 (b) shows how the high reconfiguration delay necessitates configuration prefetch to improve the latency.

We can now formulate our problem as:
For a precedence-constrained chain with some data-parallel tasks, we want to compute both the **number** of copies for each data-parallel task, and, the **workload** (execution time), t_i^j, of each (j-th) copy of such tasks, $\sum_j (t_i^j) = t_i$. Our goal is to maximize performance (minimize schedule length) given that only a fixed number of contiguous columns are available for mapping the task chain. In this work, we make an additional assumption that sufficient communication bandwidth is available for satisfying the data requirements of the multiple copies of a task.

5. PROPOSED APPROACH

In this section, we first present MFF, a heuristic for scheduling simple task chains. While MFF is oblivious to data-parallelism, it provides the core concepts underlying PARLGRAN, our proposed approach for chains with data-parallel tasks.

5.1 MFF (modified first fit)

For architectures with partial RTR, the physical (placement) constraints and, the architectural constraint of the single reconfiguration mechanism, make it difficult to achieve the ideal schedule length $L_{ideal} = \sum_{i=1}^{n} (t_i)$. In fact, this *simple* problem of minimizing schedule length for a chain, under constraints related to partial RTR, is actually NP-complete, as proved in [15]. MFF, our proposed heuristic to solve this problem, essentially tries to satisfy task resource constraints, and, attempts simple local optimizations to *reduce fragmentation*, and, hence, the schedule length.

Approach: **MFF (modified first-fit)**
Place task T_1 starting from leftmost column
for each task (T_i, i > 1)
  F_i^S = earliest time-slot enough space is available (last-fit)
  F_i^R = earliest time-slot reconfiguration controller is available
  R_i^{start} = MAX (F_i^S, F_i^R)
  E_i^{start} = MAX ($R_i^{start} + r_i$, E_{i-1}^{end})
  if (T_i aligned with rightmost column)
    local optimization: Adjust immediate ancestor placement
    (and start time) if possible to improve start time of T_i
endfor

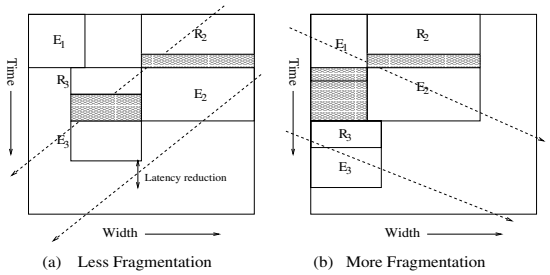


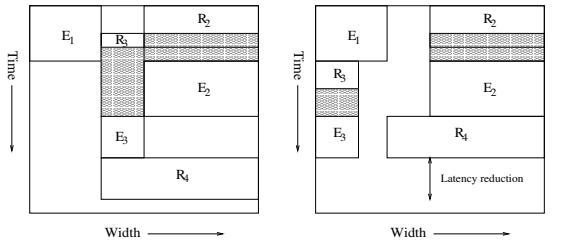
Figure 6: Simple chain- right placement of task 2

MFF is based on a first-fit approach. To get an idea why a first-fit approach works well in practical scenarios, we take a look at Figure 6 (a). The tasks are essentially laid out in the form of diagonals running from the top-right of the placed schedule towards the bottom-left. As long as a task does not "fall off" the diagonal, it is possible to overlap at least part of the reconfiguration overhead with the execution of its immediate ancestor. Once a task "falls off" the diagonal and is placed at the rightmost column C, it is essentially trying to reuse the area of ancestor tasks higher up in the chain. Given that for tasks in a chain the execution components have to be in sequence, a more distant ancestor is guaranteed to finish earlier than a closer ancestor. This increases significantly the possibility of being able to overlap reconfiguration of this task with the execution of ancestors that are closer to it in the chain. Effectively the chain property causes a "window" of tasks: tasks within a window affect each other much more strongly than tasks outside the window.

Simple fragmentation reduction: One minor modification for reducing fragmention in MFF compared to pure first-fit is shown in Figure 6. Our observations indicate that in tightly-constrained scenarios (few columns available for task mapping), placing the second task T_2 adjacent to task T_1, as in Figure 6 (b), often leads to immediate fragmentation– though enough area is available to reconfigure task T_3 in parallel with execution of task T_2, this area is not contiguous, and thus task T_3 gets delayed. MFF takes care of this by placing T_2 at the right-hand corner. Of course, this simple modification is not applicable to all scenarios.

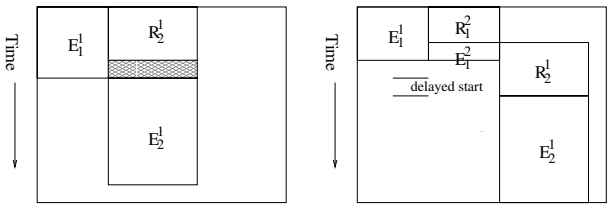
Local optimization: Exploiting slack in reconfiguration controller: A more interesting local optimization to reduce fragmentation is shown in Figure 7 (a). While scheduling task T_4, we notice that it is possible to exploit slack in the reconfiguration mechanism to *postpone* the reconfiguration R_3 of task T_3 without delaying the actual execution E_3 of task T_3. We can thus make better use of the available area (HW resources) to reschedule (and change placement of) task T_3 – as a result, reconfiguration R_4 of task T_4 can now execute in parallel with E_3, leading to a reduction in schedule length, as shown in Figure 7 (b).

Before proceeding to PARLGRAN, it is important to understand that the fragmentation problems we try to address in MFF (and



(a) More Fragmentation (b) Less Fragmentation

Figure 7: Exploiting slack in reconfiguration controller



(a) (b)

Figure 8: Static pruning based on timing

PARLGRAN) are because we are trying to jointly schedule and place while satisfying a host of other constraints– thus, other free space coalescing techniques for partially reconfigurable architectures, such as [6], are not directly applicable.

5.2 PARLGRAN

We use the insights obtained from the chain-scheduling problem as the basis for our granularity selection approach. Detailed analysis of chain-scheduling shows that applying local optimizations can improve the performance. We additionally want to design an approach such that the algorithm execution time is comparable to the execution time of the tasks. So, our proposed algorithm is simple and greedy, but, uses specific problem properties to try and improve the solution quality.

Our approach consists of two steps:

- Static pruning
- Dynamic granularity selection

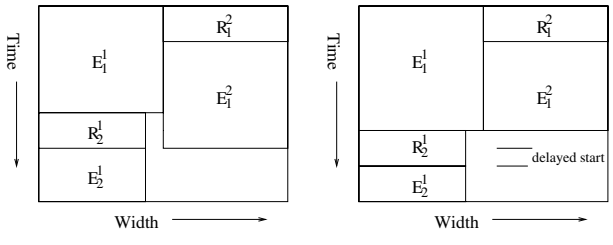
5.2.1 Static pruning

First, we utilize some simple facts to statically prune regions of the search space. As an example of pruning, consider Figure 8. If we schedule exactly one copy each for tasks T_1 and T_2, then task T_2 can start as soon as T_1 ends, i.e., at t_1, as in Figure 8 (a). If we schedule another copy of task T_1, the execution time of T_1 improves. However, now the reconfiguration controller becomes the bottleneck, as shown in Figure 8 (b). Now, task T_2 can start only at $(r_1 + r_2)$, which is greater than t_1. In general, the number of copies of a task is limited by the impact of its reconfiguration overhead on its successors.

5.2.2 Dynamic granularity selection

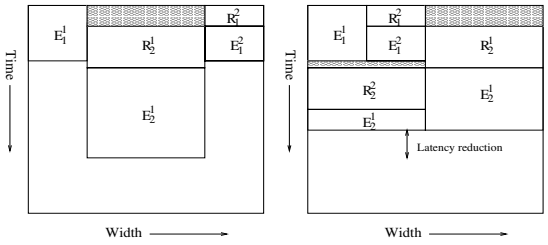
We next consider work distribution (load balancing) issues for the multiple task copies.

Uneven finish times: From our initial discussion on data-parallelism (as shown earlier in Figure 4), it seems that it is a good idea to always generate as many copies as possible subject to performance improvement and get them to finish at the same time instant. However, with the introduction of task dependencies, it is possible to modify this approach in certain cases to improve performance, as



(a) (b)

Figure 9: Uneven finish times



(a) More Fragmentation (b) Less Fragmentation

Figure 10: Left placement for copies of first task

shown in Figure 9. In Figure 9 (a) let FT_1^1 denote the time instant the earlier copy of task T_1, that is E_1^1 ends. Task T_2 can start at: $ST_2^1 = FT_1^1 + r_2$. However, if both copies of T_1 end at the same time instant as shown in Figure 9 (b), this time-instant is given by:

$$FT_1^{equal} = FT_1^1 + r_2/2$$

As a result, reconfiguration R_2 for task T_2 gets delayed and execution E_2 for task T_2 can only start at

$$FT_1^{equal} + r_2 \;=\; FT_1^1 + 3 * r_2/2$$

Of course, if the area of task T_2 is greater than the area of task T_1, letting both copies of T_1 end at the same time instant would lead to a shorter schedule.

One other minor observation to improve MFF specifically for parallelism granularity selection is shown in Figure 10. Placing multiple copies of a task adjacent to each other intuitively helps reduce fragmentation.

PARLGRAN is an adaptation of MFF that essentially tries to greedily add multiple copies of data parallel tasks as long as it estimates that adding a new copy is beneficial for performance (shorter schedule length). The concepts of dynamically adjusting the workload combined with local optimizations makes it effective. We summarize our PARLGRAN approach below.

Approach: **PARLGRAN (Parallelism Granularity Selection)**
Place first copy of task T_1 starting from leftmost column
for each task (T_i, i $>$ 1)
  Compute earliest execution start of task (space search by last-fit)
  if (parent task is data-parallel)
    while (no degradation in start time of T_i)
      add new copy of parent (assign start time, physical location)
      adjust workload of existing scheduled copies of parent
  Schedule (and place) T_i
  apply local optimizations from MFF for improving schedule
endfor

While this approach appears to be simplistic, experimental results in the following section show it typically does better than statically deciding to parallelize each task to its maximum degree. For applications like JPEG encoding, blind parallelization can lead to *significantly inferior* results, even worse than RTR-aware first-fit,

because of the reconfiguration overhead and the physical (placement) constraints.

6. EXPERIMENTS

We conducted a large set of experiments with over a 1000 datapoints to demonstrate the quality of schedules generated by our heuristics. In this section we present the key results from our experiments. It is important to remember that our goal is to maximize the performance (minimize schedule length) of a chain of tasks given a hard constraint on the available area. Therefore, while it is possible to fit our applications onto suitably sized target devices, we assume for experimental purposes that the resource constraint is less than the aggregate size of all tasks. This makes our approach suitable for a scenario where multiple processes are executing simultaneously on the reconfigurable device and the area available for mapping the task graph is not known beforehand.

Experimental setup

We assumed a target device organized as a CLB matrix of 56 rows, 48 columns, similar to Xilinx XC2V2000. From the XC2V2000 data sheet, we estimate that the reconfiguration overhead for the smallest task occupying one column on our architecture is 0.19 ms at the maximum suggested reconfiguration frequency of 66 MHz. We obtained area and timing data for tasks like Huffman, DCT, etc., by synthesizing tasks under columnar placement and routing constraints on the XC2V2000, similar to the Xilinx methodology suggested for "reconfigurable modules".

We explored a large set of possible scenarios by generating task chains which varied the following parameters: (1) varying chain length, in the range of 4 to 20 tasks in the chain. (2) varying task parameters for each task in a chain of given length (3) varying area constraints.

That is, for each chain length in the given range (4 to 20), we had a set of testcases where the tasks had different parameters. For each such testcase, we varied the area constraint across a wide range to represent loose as well as tight constraints, thus generating data for over a thousand individual experiments.

The task parameters (execution time, reconfiguration delay, number of columns) were randomly selected from our database of synthesized tasks. The database consisted of information corresponding to images of various sizes- since each task is completely pipelined, the reconfiguration delay and number of columns occupied by the task is independent of the image size, but, the execution time is directly proportional to the image size.

We measure schedule quality of our proposed heuristic by comparing against two approaches: FF (first-fit) and MAXPARL (maximum parallelization). MAXPARL attempts to maximize parallelization by statically generating the maximum number of copies possible for each task subject to resource constraints only, and assigns equal workload to each copy. In subsequent discussions, the following notation denotes schedule lengths generated by the various approaches on an individual experiment:

L_{ff}: corresponds to first-fit approach

L_{max}: corresponds to MAXPARL (maximum parallelization) approach

L_{mff}: corresponds to our proposed MFF approach

L_{gran}: corresponds to our proposed PARLGRAN (parallelism granularity selection) approach

Schedule quality of MFF

Our first set of experiments consisted of comparing schedule lengths generated by MFF with that of first-fit, on the set of experiments as described above.

The experimental data confirmed that schedules generated by MFF were almost always equal to or better than first-fit. The schedule lengths generated by MFF were better in 243 out of 1140 tests, i.e., approximately 20% of the tests, worse in 5 out of 1140 tests. In 113 tests, around 10% of the total, MFF was better by at least 3%. In the worst experiment for MFF, first-fit generated a schedule longer by 0.44%. Overall, on longer chains (more tasks) and looser constraints (more columns), both algorithms were almost equally able to hide the reconfiguration overhead. However, on more constrained problems with shorter chains and tighter area constraints, MFF tends to generate better schedules.

Chain length	PARLGRAN Vs FF	PARLGRAN Vs MAXPARL		
	Avg	Avg	Best	Worst
4-7	46.3%	9.8%	142.5%	-49.6%
8-11	51.7%	15.8%	109.6%	-30.9%
12-15	55.0%	18.5%	82.3%	-15.5%
16-20	58.3%	33.8%	151%	-17.5%
Avg gain	>50%	> 15%		

Table 1: Reduction in schedule length for completely data parallel chains with PARLGRAN

Schedule quality of PRLGRAN

Next, in Table 1, we present a summary of results conducted on our PARLGRAN (parallelism granularity selection) approach. The data in each row of the table corresponds to experiments on chains of corresponding length– as an example, data in row 2 (chain length 8-11) was obtained from experiments on chains with at least 8 tasks and at most 11 tasks. Note that this set of experiments is identical to that we used to validate MFF– the difference is that we now assume each task in the chain is completely data-parallel. For comparison with MAXPARL and FF, our quality measure is simply the percentage increase in schedule length generated by the other approach compared to PARLGRAN.. As an example, for comparison with MAXPARL, the quality measure is simply:

$((L_{max} - L_{gran})/L_{gran}) * 100$

The first column in Table 1 represents the *Average* percentage improvement of PARLGRAN as compared to FF. Each entry in the first column is an average of a large number of experiments conducted on chains of corresponding length. The second, third and fourth columns respectively represent the *Average*, the *Best* and the *Worst* performance of our approach compared to MAXPARL. As an example, the data in row 2, column 3, states that on a large number of chains with chain length between 8 and 11 tasks, the best result generated by our approach corresponds to an experiment where MAXPARL generated a schedule 109% longer.

The table clearly shows that our 'granularity selection' heuristic, PARLGRAN, generates increasingly better results compared to MAXPARL when more space is available. Intuitively, with more space, it is possible to make more instances of the data-parallel tasks. However, with each additional instance, the workload (execution time) decreases per instance, making the execution time comparable to the reconfiguration overhead – PARLGRAN is much better capable of deciding when to stop instantiating multiple copies, as opposed to MAXPARL. The local optimizations in PARLGRAN play an active role in such circumstances to help improve the schedule length. One key experimental aspect we would like to mention is that for smaller chains, our presented results cover a very large range of varying area constraints– for the longer chains, the presented results cover the scenarios where the available HW area is at most around 40% of the aggregate HW area of the tasks. More detailed results in [16] confirm that the benefit of our approach over MAXPARL increases significantly as chain length gets longer and available area increases.

Case	C	L_{mff} (ms)	L_{max} (ms)	L_{gran} (ms)
256X256 JPG	5	12.71	12.73	**12.36**
	6	11.24	12.52	**10.81**
	7	11.24	11.38	**10.05**
	8	11.24	12.11	**9.08**
	9	10.10	12.79	**9.08**
512X512 JPG	5	42.86	40.68	40.30
	6	41.34	35.32	35.13
	7	41.34	34.18	34.37
	8	41.34	29.08	28.60
	9	40.20	28.38	27.71

Table 2: Case study of JPEG encoding: Schedule Length with different image size and area constraints

Case study of JPEG encoding

After conducting a wide range of experiments on synthetic graphs, we conducted a case study on the JPEG encoding algorithm, represented as a chain of four key tasks: RGB2YCbCr, DCT, Quantize, Huffman. Table 2 presents the consolidated results from the case study. Entries in the first column, CASE, denote the image size – 256X256 denotes experiments on a 256X256 colour image. For each case, we varied the number of columns and observed the resulting schedule lengths. The second column C represents the area constraint in columns. The third, fourth and fifth columns correspond to schedule lengths (in ms) generated by MFF, MAXPARL, and PARLGRAN respectively.

For the 256X256 image, the reconfiguration overheads are comparable to the task execution times. Our approach frequently does much better than statically parallelizing everything, as in MAXPARL– the results confirm that such blind parallelization can often lead to results worse than a simple sequential scheduling approach. For an area constraint of 8 columns, schedule length of FF is longer than PARLGRAN by (11.24-9.08)/9.08 = 23.5%. Blind (static) parallelization leads to significantly worse schedule longer by (12.11-9.08)/9.08 = 33.3%. This is in spite of the fact that the effective transformed graph from MAXPARL consisted of 9 tasks with apparently more parallelism, while the transformed graph from PARLGRAN consisted of 7 tasks only.

For the 512X512 image, each task execution time is significantly greater than the reconfiguration overhead. In such a scenario, where, additionally, the chain length is short, MAXPARL generates good results – of course, PARLGRAN typically does somewhat better. But, both parallelizing approaches result in significant speedups.

Estimated run-time: Preliminary estimates indicate that the run-time of our approach on a PowerPC processor at 400 MHz (available on the Virtex-II Pro platform from Xilinx) is around 3-4 ms for a reasonably large experiment with 12 tasks and 20 columns. Considering the fact that the run-time of the DCT task for 512X512 colour image is around 11 ms on the target architecture, our approach is suitable for *semi-online* scenarios where the task precedence relations, and the task area-timing characteristics are available at compile-time, while the available HW area for for mapping a DAG is known only at run-time. Task management under such dynamic resource availability is a key issue in modern operating systems for reconfigurable architectures [5].

Our wide range of experiments and case studies confirm that PARLGRAN generates high-quality schedule in all situations– tightly constrained problems with shorter chains, fewer columns, as well as problems with more degrees of freedom, i.e., longer chains, more available columns. Additionally, the estimated run-time of our approach on a typical embedded processor is comparable to the HW task execution times.

7. CONCLUSION

In this paper, we proposed PARLGRAN, an approach that selects granularity of data-parallelism to maximize performance of application *task chains* executing on an architecture with partial RTR (run-time reconfiguration). Our approach selects both the number of instances of a data-parallel task, and, the execution time of each such instance – it is integrated in a joint scheduling and placement formulation, necessitated by the underlying physical and architectural constraints imposed by partial RTR. Experimental results on a significantly large space of over a 1000 synthetic experiments confirm that our approach generates schedules that are on an average better by 15% over an approach that tries to *statically* maximize data-parallelism. A detailed case study on JPEG encoding confirms that in realistic scenarios, an approach that simply tries to maximize data parallelism without accounting for the underlying constraints can end up generating schedules *much worse* than even a data-parallelism-oblivious (but RTR-aware) approach. Initial estimates indicate PARLGRAN is fast enough to be suitable for integration in a *semi-online* scheduling methodology where the goal is to maximize performance of an application given an area constraint known only at run-time.

While our approach demonstrates the potential for significant performance improvement, there are some key aspects that we want to address in our future work. First, we have assumed in this work that we are not constrained by memory/communication bandwidth. As we increase the task granularity (make more instances to exploit more data-parallelism), the data transfer to and from memory, both on-chip, and, off-chip, has the potential to become a bottleneck and will be considered in future work. We also plan to study the performance versus energy characteristics of such implementations on reconfigurable architectures with partial RTR.

8. REFERENCES

[1] S Banerjee, E Bozorgzadeh, N Dutt, "Physically-aware HW-SW Partitioning for reconfigurable architectures with partial dynamic reconfiguration", DAC, 2005

[2] J Harkin, T M Mcginnity, L P Maguire, "Modeling and Optimizing Run-Time reconfiguration using evolutionary computation", ACM TECS, V-3, Nov 2004

[3] P-H Yuh, C-L Yang, Y-W Chang,H-L Chen, "Temporal floorplanning using the T-tree formulation", ICCAD, 2004

[4] J Noguera, R M Badia, "Power-Performance trade-offs for reconfigurable computing", CODES+ISSS, 2004

[5] C Steiger, H Walder, M Platzner, "Operating systems for reconfigurable embedded platforms: Online Scheduling of Real-Time Tasks", IEEE Trans on Computers, V-53, 11, Nov 2004

[6] M Handa, R Vemuri, "An efficient algorithm for finding empty space for online FPGA placement", DAC, 2004

[7] H Quinn, L A Smith King, M Leeser, W Meleis, "Runtime Assignment of Reconfigurable Hardware Components for Image Processing Pipelines", FCCM, 2003

[8] S P Fekete, E Kohler, J Teich, "Optimal FPGA module placement with temporal precedence constraints", DATE, 2001

[9] H Singh, G Lu, E M C Filho, R Maestre, M-H Lee, F J Kurdahi, N Bagherzadeh, "MorphoSys: case study of a reconfigurable computing system targeting multimedia applications", DAC, 2000

[10] S Hauck, "Configuration pre-fetch for single context reconfigurable processors", FPGA, 1998

[11] M J Wirthlin, "Improving functional density through Run-time Circuit Reconfiguration", PhD Thesis, Electrical and Computer Engineering Dept, Brigham Young Univesity, 1997

[12] G Brebner, "A virtual hardware operating system for the Xilinx XC6200", FPL, 1996

[13] H Murata, K Fujiyoshi, S Nakatake, Y Kajitani, "Rectangle-packing based module placement", ICCAD, 1995

[14] S Muchnick, "Advanced Compiler design and implementation", Morgan Kaufmann, 1997

[15] J Augustine, Personal communication

[16] S Banerjee, E Bozorgzadeh, N Dutt, "Selecting granularity of parallelism for tasks executing on dynamically reconfigurable architectures", CECS Technical Report, UC Irvine.