A Pattern Selection Algorithm for Multi-Pattern Scheduling

Yuanqing Guo Cornelis Hoede Gerard J.M. Smit
Faculty of EEMCS, University of Twente
P.O. Box 217, 7500AE Enschede, The Netherlands
E-mail: {y.guo, c.hoede, g.j.m.smit}@utwente.nl

Abstract

The multi-pattern scheduling algorithm is designed to schedule a graph onto a coarse-grained reconfigurable architecture, the result of which depends highly on the used patterns. This paper presents a method to select a near-optimal set of patterns. By using these patterns, the multi-pattern scheduling will result in a better schedule in the sense that the schedule will have fewer clock cycles.

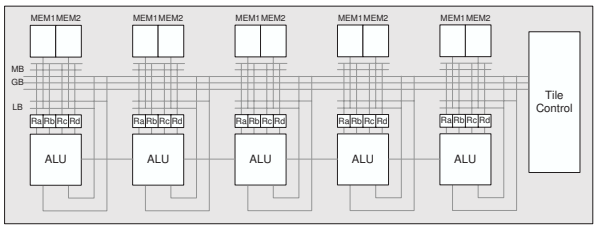

Figure 1. Montium processor tile

1 Introduction

The most commonly used computer system architectures in data processing nowadays can be divided into three categories: General Purpose Processors (GPPs), application specific architectures and reconfigurable architectures. GPPs are flexible, but inefficient and for some applications with not enough performance. Application specific architectures are efficient and with good performance, but inflexible. Recently reconfigurable systems have drawn more and more attention due to their combination of flexibility and efficiency. Reconfigurable architectures limit their flexibility to a particular algorithm domain. A Montium tile [2] is a coarse-grained reconfigurable system (see Figure 1), designed at the University of Twente. In the Montium, the functions of Arithmetic and Logic Units (ALUs) can be changed by reconfigurations. One Montium tile has five ALUs which, for instance, can be configured to compute two additions and three multiplications during the first clock cycle, and one addition, two subtractions and two bit-or operations during the second clock cycle. The combination of concurrent functions that can be performed on the parallel reconfigurable ALUs in one clock cycle is called *a pattern*.

The programmability of reconfigurable architectures differs considerably from that of GPPs. In a GPP, the ALU can be programmed many times and to any of its possible functions. However, in the Montium, although the five ALUs can execute thousands of different possible patterns, or efficiency reasons during one application, it is only allowed to use up to 32 of them.

To automate the design process and achieve optimal exploitation of the architectural features of the Montium, a high level entry compiler for the MONTIUM architecture is currently being implemented [3]. This approach consists of four phases: Transformation, Clustering, Scheduling and Allocation. In this paper, we only concentrate on the scheduling phase.

In [1], a multi-pattern scheduling algorithm is presented, which is to schedule the nodes of a graph assuming that a fixed number P_{def} of patterns are given. The experimental results of the multi-pattern scheduling algorithm showed that it is very sensitive to the selected patterns. In this paper we present a method to choose P_{def} patterns.

The rest of the paper is organized as follows: Some related work is given in Section 2; In Section 3, some definitions are given; Since the result of the pattern selection algorithm is used by the multi-pattern scheduling, we will give a short description of the latter in Section 4; The proposed pattern selection algorithm is described in Section 5; Finally the experimental results

1-4244-0054-6/06/$20.00

and conclusions are presented in Section 6 and Section 7.

2 Related work

Scheduling is a well defined and studied problem in the research area of high-level synthesis [4]. Most scheduling problems are NP-complete problems [5]. To solve the scheduling problems heuristic algorithms have been used to find feasible (possibly sub-optimal) solutions. Two commonly used heuristic algorithms are: list scheduling [6][7] and force-directed scheduling[8].

As far as we know, none of the existing scheduling methods can be used for the Montium, a coarse-grained reconfigurable architecture. The number of patterns is restricted for the scheduling problem in the Montium, which has never been considered in the traditional scheduling methods.

3 Definitions

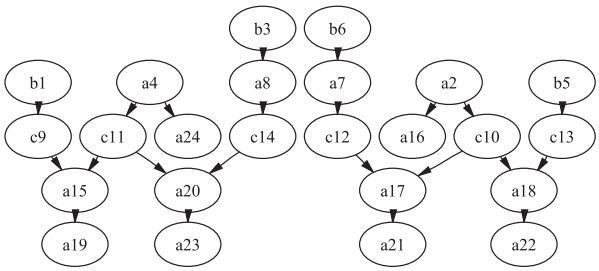

Figure 2. Multi-pattern scheduling example: 3DFT algorithm

On a Data Flow Graph (DFG) a node n represents a function/operation and a directed edge denotes a dependency between two operations. If there is an edge directing from node n_1 to n_2, n_1 is called a *predecessor* of n_2, and n_2 is called a *successor* of n_1. $Pred(n)$ represents the set formed by all the predecessors of node n and $Succ(n)$ represents the set formed by all the successors of node n. We call n a *follower* of m, if there exists a sequence $n_0, \ldots, n_k$ of nodes such that $n_0 = m$, $n_k = n$, and n_i is a predecessor of n_{i+1} for all $i \in \{0, \ldots, k-1\}$.

The As Soon As Possible level ($ASAP(n)$) attribute indicates the earliest time that the node n may be scheduled. It is computed as:

$$ASAP(n) = \begin{cases} 0 & \text{if } Pred(n) = \phi; \\ \max\limits_{\forall n_i \in Pred(n)} (ASAP(n_i) + 1) & otherwise. \end{cases} \quad (1)$$

The As Late As Possible level ($ALAP(n)$) attribute determines the latest clock cycle the node n may be scheduled. It is computed using the following:

$$ALAP(n) = \begin{cases} ASAP_{max} & \text{if } Succ(n) = \phi; \\ \min\limits_{\forall n_i \in Succ(n)} (ALAP(n_i) - 1) & otherwise, \end{cases} \quad (2)$$

where $ASAP_{max} = \max\limits_{\forall n_i \in \mathcal{N}} (ASAP(n_i))$.

The Height ($Height(n)$) of a node is the maximum distance between this node and a node without successors. It is calculated as follows:

$$Height(n) = \begin{cases} 1 & \text{if } Succ(n) = \phi; \\ \max\limits_{\forall n_i \in Succ(n)} (Height(n_i) + 1) & otherwise. \end{cases} \quad (3)$$

The ASAP level, ALAP level and Height of all nodes are listed in TABLE 1.

Table 1. ASAP level, ALAP level and Height

	asap	alap	Height		asap	alap	Height
b3	0	0	5	b6	0	0	5
b1	0	1	4	b5	0	1	4
a4	0	1	4	a2	0	1	4
a8	1	1	4	a7	1	1	4
c9	1	2	3	c13	1	2	3
c11	1	2	3	c10	1	2	3
a24	1	4	1	a16	1	4	1
a15	2	3	2	a18	2	3	2
a20	3	3	2	a17	3	3	2
a19	3	4	1	a22	3	4	1
a23	4	4	1	a21	4	4	1

The type of the function of a node n is called a *color* of n, written as $l(n)$. The scheduling objective is to associate each node of a DFG to a clock cycle such that certain constraints are met.

In a system with a fixed number (denoted by C, which is 5 in the Montium architecture) of reconfigurable resources, C functions that can be run by the C reconfigurable resources in parallel are called a pattern. A pattern is therefore a bag [1] of C elements. A

[1] A bag, or multi-set, is an unordered collection of values that may have duplicates.

pattern might have less than C colors. The undefined elements are represented by dummies.

On a DFG, two nodes n_1 and n_2 are called *parallelizable* if neither n_1 is a follower of n_2 nor n_2 is a follower of n_1. If $\mathcal{A}$ is a set of one node or pairwise parallelizable nodes we say that $\mathcal{A}$ is an *antichain* (this concept is borrowed from the theory of posets, i.e. partially ordered sets. (Please refer to [9] for more information.) If the size of an antichain $\mathcal{A}$ is smaller than or equal to C, we say that $\mathcal{A}$ is *executable*. In Fig. 2, set $\mathcal{A}1 = \{\text{b1}, \text{a4}, \text{b3}, \text{b6}, \text{a16}, \text{c10}\}$ is an antichain, while $\mathcal{A}2 = \{\text{b1}, \text{a4}, \text{b3}, \text{b6}, \text{a16}, \text{a17}\}$ is not because a17 is a follower of b6. When $C = 5$, $\mathcal{A}1$ is not executable and $\mathcal{A}3 = \{\text{b1}, \text{a4}, \text{b3}, \text{b6}, \text{a16}\}$ is executable.

4 A multi-pattern scheduling algorithm

1. Compute the priority function for each node in the graph.
2. Get the candidate list.
3. Sort the nodes in the candidate list according to their priority functions.
4. Schedule the nodes in the candidate list from high priority to low priority according to all given patterns.
5. Compute the pattern priority function for each pattern and keep the pattern with highest pattern priority value.
6. Update the candidate list.
7. If the candidate list is not empty, go back to 3; else end the program.

Figure 3. Multi-Pattern List Scheduling Algorithm

Given a set of patterns $\bar{p}_1, \bar{p}_2, \cdots \bar{p}_{P_{def}}$, the objective of the multi-pattern scheduling problem is to assign nodes of a DFG to a clock cycle such that (1) the dependencies between nodes are satisfied, (2) within each clock cycle the needed resources are determined by the resources defined by one of the given patterns and, (3) the number of clock cycles is minimized.

A list based algorithm maintains a *candidate list* CL of *candidate nodes*, i.e., nodes whose predecessors have already been scheduled. The candidate list is sorted according to a priority function of these nodes. In each iteration, nodes with higher priority are scheduled first and lower priority nodes are deferred to a later clock cycle. Scheduling a node within a clock cycle makes its successor nodes candidates, which will then be added to the candidate list.

For multi-pattern scheduling, for one clock cycle, not only nodes but also a pattern should be selected. The selected nodes should not use more resources than the resources presented in the selected pattern. For a specific candidate list CL and a pattern $\vec{p}_i$, a *selected set* $S(\vec{p}_i, CL)$ is defined as the set of nodes from CL that will be scheduled provided the resources are given by $\vec{p}_i$.

The multi-pattern scheduling algorithm is given in Fig. 3. In total two types of priority functions are defined here, the *node priority* and the *pattern priority*. The former is for each node in the graph and the latter is for scheduling elements from a candidate list by one specific pattern.

4.1 Node priority

In the algorithm, the following priority function for graph nodes is used:

$$\begin{aligned} f(n) &= s \times height + t \times \#\text{direct\_successors} \\ &\quad +\#\text{all\_successors} \end{aligned} \tag{4}$$

Here #direct_successors is the number of the successors that follow the node directly, and #all_successors is the number of all successors. Parameters s and t are used to distinguish the importance of the factors. s and t should satisfy the following conditions:

$$\begin{aligned} s &\geq \max\{t \times \#\text{direct\_successors} \\ &\qquad +\#\text{all\_successors}\} \\ t &\geq \max\{\#\text{all\_successors.}\} \end{aligned} \tag{5}$$

These conditions guarantee that the node with largest height will always have the highest priority; For the nodes with the same height, the one with more direct successors will have higher priority; For the nodes with both the same height and the same number of direct successors, the one with highest number of successors will have highest priority.

The height of a node reflects its scheduling flexibility. For a given candidate list, the node with smaller height is more flexible in the sense that it might be scheduled at a later clock cycle. Nodes with largest height are given the preference to be scheduled earlier. The scheduling of the nodes with more direct successors will make more nodes go to the candidate list, they are therefore given higher priority. Furthermore, the node with more successors is given higher priority since the delaying of the scheduling of this node will cause the delaying of the scheduling of more successors.

4.2 Pattern priority

Intuitively for each clock cycle we want to choose the pattern that can cover most nodes in the candidate list.

Table 2. Scheduling Procedure

clock cycle	candidate list	pattern1 ="aabcc"	pattern2 ="aaacc"	selected pattern
1	a2,a4,b1,b3,b5,b6	a2,a4,b6	a2,a4	*1*
2	b1,b3,b5,c11,a24, a16,c10,a7	a7,a24,b3,c10, c11	a24,a16,a7,c11, c10	*1*
3	a8,a16,b1,b5,c12	a8,a16,b5,c12	a8,a16,c12	*1*
4	b1,c14,a17,c13	a17,b1,c13,c14	a17,c13,c14	*1*
5	a18,a20,a21,c9	a18,a20,c9	a18,a20,a21,c9	*2*
6	a15,a22,a23	a15,a22	a15,a22,a23	*2*
7	a19	a19	a19	*1*

This leads to a definition of the priority function for a pattern $\vec{p}$ corresponding to a candidate list CL.

$$F_1(\vec{p}, CL) \quad = \text{number of nodes in selected set } S(\vec{p}, CL). \tag{6}$$

On the other hand, the nodes with higher priorities should be scheduled before those with lower priorities. That means that we prefer the pattern that covers more high priority nodes. Thus we modify the priority of a pattern as the sum of priorities of all nodes in the selected set.

$$F_2(\vec{p}, CL) = \sum_{n \in S(\vec{p}, CL)} f(n). \tag{7}$$

4.3 Example

We explain the algorithm with the help of the 3-point Fast Fourier Transform (3DFT) algorithm. The DFG of 3DFT consists of additions, subtractions and multiplications, as shown in Fig. 2. The first letter of the name of a node is the color of the node. The nodes denoted by "a" are additions; while those with "b" represent subtractions and the nodes with "c" multiplications. Two patterns are assumed to be given here: pattern1 = "aabcc" and pattern2 = "aaacc". The scheduling procedure is shown in TABLE 2. Initially, there are six candidates: {a2, a4, b1, b3, b5, b6}. If we use pattern1 {a2, a4, b6} will be scheduled, and if we use pattern2 {a2, a4} will be scheduled. Because the priority function of pattern1 is larger than that of pattern2, pattern1 is selected. For the second clock cycle, pattern1 covers nodes {a7, a24, b3, c10, c11} while pattern2 covers {a7, a16, a24, c10, c11}. The difference between the use of the two patterns lies in the difference between b3 and a16. If we use the pattern priority function $F_1(\vec{p}, CL)$ defined in Equation (6), the two patterns are equally good. The algorithm will pick one at random. If we use $F_2(\vec{p}, CL)$ defined in Equation (7) as pattern priority function, pattern1 will be chosen because the height of b3 is larger than that of a16.

4.4 Experiment

Table 3. Experimental results: Number of clock cycles for the final scheduling

patterns	clock cycles
{a,b,c,b,c}, {b,b,b,a,b}, {b,b,b,c,b}, {b,a,b,a,a}	8
{a,b,c,b,c}, {b,c,b,c,a}, {c,b,a,b,a}, {b,b,c,c,b}	9
{a,b,c,c,c}, {a,a,b,a,c}, {c,c,c,a,a}, {a,b,a,b,b}	7

We ran the multi-pattern scheduling algorithm on the 3-Fast Fourier Transform (3DFT) algorithms by using 4 patterns. The experimental results are given in TABLE 3 where the number indicates the number of clock cycles needed. From the experiment we can see that: The selection of patterns has a very strong influence on the scheduling results!

5 Pattern selection

We saw in the previous section that the selection of patterns is very important. In this section we present a method to choose P_{def} patterns.

The requirements to the selected patterns are:

1. The selected patterns cover all the colors that appear in the DFG;
2. The selected patterns appear frequently in the DFG (have many antichains in the DFG).

Our proposed method first finds all the possible patterns and their antichains in the DFG presented in Section 5.1, and then makes the selection from them represented in Section 5.2.

5.1 Pattern generation

The pattern generation method finds all antichains of size C first and then the antichains are classified according to their patterns as follows:

pattern1: antichain11,antichain12,antichain13,$\cdots$
pattern2: antichain21,antichain22,antichain23,$\cdots$
pattern3: antichain31,antichain32,antichain33,$\cdots$
$\vdots$

In the small example shown in Fig. 4, the same as the example in Fig. 2, the letters "a" and "b" represent the colors. The classified antichains are listed in TABLE 4:

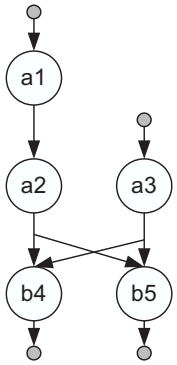

Figure 4. A small example for the pattern selection algorithm

Table 4. Patterns and antichains in the DFG in Fig. 4

patterns	antichains
$\bar{p}_1 = \{a\}$:	{a1},{a2},{a3}
$\bar{p}_2 = \{b\}$:	{b4},{b5}
$\bar{p}_3 = \{aa\}$:	{a1,a3},{a2,a3}
$\bar{p}_4 = \{bb\}$:	{b4,b5}

The number of antichains increases very fast with the size. The elements of an antichain may haven been chosen from different levels of the DFG. The concept *span* for an antichain $\mathcal{A}$ captures the difference in level, which is defined as follows:

$$Span(\mathcal{A}) = U(\max_{n\in\mathcal{A}}\{ASAP(n)\} - \min_{n\in\mathcal{A}}\{ALAP(n)\}),$$

where, $U(x)$ is a function defined as follows:

$$U(x) = \begin{cases} 0 & x < 0; \\ x & x \geq 0. \end{cases}$$

Looking at an antichain $\mathcal{A} = \{a24, b3\}$ in Fig. 2, the levels of the nodes are: $ASAP(\text{a24}) = 1$, $ALAP(\text{a24}) = 4$, $ASAP(\text{b3}) = 0$ and $ALAP(\text{b3}) = 0$. Therefore,

$$\max_{n\in\mathcal{A}}\{ASAP(n)\} = \max\{1, 0\} = 1;$$

$$\min_{n\in\mathcal{A}}\{ALAP(n)\} = \min\{0, 4\} = 0.$$

The span is

$$Span(\mathcal{A}) = U(1 - 0) = 1.$$

Theorem 1 *If the nodes of an antichain $\mathcal{A}$ are scheduled in one clock cycle, the total number of clock cycles of the final schedule will be at least $ASAP_{max} + Span(\mathcal{A}) + 1$.*

Proof Assume that node $n1$ has the minimal ALAP level and node $n2$ has the maximal ASAP level (see Fig. 5). Before $n2$, there are at least $ASAP(n2)$ clock cycles and after $n1$, there are at least $ASAP_{max} - ALAP(n1)$ clock cycles. If $n1$ and $n2$ are run at the same clock cycle, when $ASAP(n2)$ is larger than $ALAP(n1)$ as is the case in Fig. 5, totally at least $ASAP(n2) + ASAP_{max} - ALAP(n1) + 1$ clock cycles are required for the whole schedule, where the extra 1 is for the clock cycle when the $n1$ and $n2$ are executed. However, the total number of clock cycles cannot be smaller than $ASAP_{max} + 1$, which is the length of the longest path on the graph. Thus when $ASAP(n2) \leq ALAP(n1)$, the length of the schedule is larger than or equal to $ASAP_{max} + 1$.

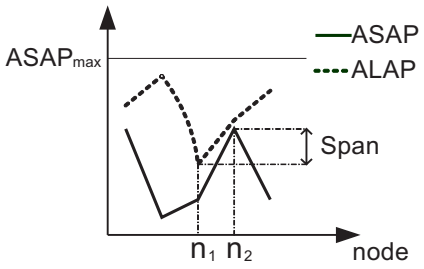

Figure 5. Span

Theorem 1 shows that to run the nodes of an $\mathcal{A}$ with too large span in parallel will decrease the performance of the scheduling. A pattern with many antichains, all of which are with very large span, is therefore not a favorable pattern. We will see soon that the antichains of a pattern will contribute to the preference to take the pattern. Due to the above analysis, it is not useful to take antichains of large span into consideration. For

instance, in the graph of Fig. 2 node "a19" and node "b3" are unlikely to be scheduled to the same clock cycle although they are parallelizable. The number of antichains decreases by setting a limitation to the span of antichains, which, on the other hand, also decreases the computational complexity (See TABLE 5 for the number of antichains for the 3DFT satisfying the span limitation).

Table 5. The number of antichains that satisfy the span limitation for 3DFT

Number of nodes in $\mathcal{A}$	1	2	3	4	5
$Span(\mathcal{A})$ =4	24	224	1034	2500	3104
$Span(\mathcal{A})$ =3	24	222	1010	2404	2954
$Span(\mathcal{A})$ =2	24	208	870	1926	2282
$Span(\mathcal{A})$ =1	24	178	632	1232	1364
$Span(\mathcal{A})$ =0	24	124	304	425	356

5.2 Pattern selection

The pseudo-code for selecting patterns is given in Fig. 6. Non-ordered patterns are selected one by one

```
1  for(i = 0; i < P_def; i++) {
2      Compute the priority function for each pattern;
3      Choose the pattern with the largest priority
       function;
4      Delete the subpatterns of the selected pattern.
5  }
```

Figure 6. The pseudo-code for pattern selection procedure

based on priority functions. The key technique is the computation of the priority function for each pattern (line 2 in Fig. 6), which is decisive for the potential use of the selected pattern. After one pattern is selected, all its subpatterns are deleted (line 3) because we can use the selected pattern at the place where a subpattern is needed.

In the multi-pattern list scheduling algorithm given in Section 4, a node which forms a pattern with other parallelizable nodes will be scheduled. If the allowed patterns for the multi-pattern list scheduling algorithm cover more antichains including a specific node, it is easier to schedule the node. The idea now is that the number of antichains of the selected patterns that cover a node should be as large as possible, and the number should be balanced among all nodes because some unscheduled nodes might decrease the performance of the scheduling.

For each pattern $\bar{p}$, a *node frequency*, $h(\bar{p}, n)$ is defined to represent the number of antichains that include a node n. The node frequencies of all nodes form an array:

$$\vec{h}(\bar{p}) = (h(\bar{p}, n_1), h(\bar{p}, n_2), \cdots, h(\bar{p}, n_N)).$$

$h(\bar{p}, n)$ tells how many different ways there are to schedule n by the pattern $\bar{p}$, or we can say that $h(\bar{p}, n)$ indicates the flexibility to schedule the node n by the pattern $\bar{p}$. The vector $\vec{h}(\bar{p})$ indicates not only the number but also the distribution of the antichains over all nodes.

Suppose t patterns have been selected and they are represented by $\mathbb{P}_s = \{\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_t\}$. The priority function of the remaining patterns for selecting the $(t+1)$th pattern is defined as:

$$f(\bar{p}_j) = \sum_{n \in \mathcal{N}} \frac{h(\bar{p}_j, n)}{\sum_{\bar{p}_i \in \mathbb{P}_s} h(\bar{p}_i, n) + \varepsilon} + \alpha \times |\bar{p}_j|^2 \quad \text{for } \bar{p}_j \notin \mathbb{P}_s. \qquad (8)$$

We want to choose the pattern that occurs more often in the DFG. Therefore the priority function is larger when a node frequency $h(\bar{p}_j, n)$ is larger. To balance the node frequencies for all nodes, $\sum_{\bar{p}_i \in \mathbb{P}_s} h(\bar{p}_i, n)$ is used, which is the number of antichains containing node n among all the selected patterns. When other patterns already have many antichains to cover node n, the effect of the node frequency in the next pattern becomes less. ε is a constant value to avoid that 0 is used as the divisor. The size of a pattern $|\bar{p}_j|$ means the number of colors in pattern $\bar{p}_j$. α is a parameter. By $\alpha \times |\bar{p}_j|^2$, larger patterns are given higher priority than smaller ones. We will see the reason in the following example. In our system $\varepsilon = 0.5$ and $\alpha = 20$.

Let us use the example in Fig. 4 to demonstrate the above mentioned algorithm. The node frequencies are given in TABLE 6.

Table 6. Node frequencies

	a1	a2	a3	b4	b5
$\bar{p}_1 = \{a\}$	1	1	1	0	0
$\bar{p}_2 = \{b\}$	0	0	0	1	1
$\bar{p}_3 = \{aa\}$	1	1	2	0	0
$\bar{p}_4 = \{bb\}$	0	0	0	1	1

At the very beginning, there is no selected pattern, i.e., $\mathbb{P}_s = \phi$. $\sum_{\bar{p}_i \in \mathbb{P}_s} h(\bar{p}_i, n)$ is therefore always zero. The

priorities are:

$$\begin{aligned} f(\bar{p}_1) &= \frac{1}{\varepsilon}+\frac{1}{\varepsilon}+\frac{1}{\varepsilon}+0+0+20\times 1^2 = 26; \\ f(\bar{p}_2) &= 0+0+0+\frac{1}{\varepsilon}+\frac{1}{\varepsilon}+20\times 1^2 = 24; \\ f(\bar{p}_3) &= \frac{1}{\varepsilon}+\frac{1}{\varepsilon}+\frac{2}{\varepsilon}+0+0+20\times 2^2 = 88; \\ f(\bar{p}_4) &= 0+0+0+\frac{1}{\varepsilon}+\frac{1}{\varepsilon}+20\times 2^2 = 84; \end{aligned}$$

Obviously $\bar{p}_3$ is the first selected pattern. Correspondingly $\bar{p}_1$ is deleted because it is a subpattern of $\bar{p}_3$. For choosing the second pattern, we have

$\sum_{\bar{p}_i\in\mathbb{P}_s} h(\bar{p}_i, \text{a1}) = 1; \sum_{\bar{p}_i\in\mathbb{P}_s} h(\bar{p}_i, \text{a2}) = 1;$
$\sum_{\bar{p}_i\in\mathbb{P}_s} h(\bar{p}_i, \text{a3}) = 2; \sum_{\bar{p}_i\in\mathbb{P}_s} h(\bar{p}_i, \text{b4}) = 0;$
$\sum_{\bar{p}_i\in\mathbb{P}_s} h(\bar{p}_i, \text{b5}) = 0.$

The priorities become:

$$\begin{aligned} f(\bar{p}_2) &= 0+0+0+\frac{1}{\varepsilon}+\frac{1}{\varepsilon}+20\times 1^2 = 24; \\ f(\bar{p}_4) &= 0+0+0+\frac{1}{\varepsilon}+\frac{1}{\varepsilon}+20\times 2^2 = 84; \end{aligned}$$

The priority functions for $\bar{p}_2$ and $\bar{p}_4$ keep the old value. The reason is that pattern $\bar{p}_3$ has antichains that cover nodes "a1", "a2" and "a3", while $\bar{p}_2$ and $\bar{p}_4$ only relate to "b4" and "b5". If there were another pattern which covered node "a1", "a2" or "a3", the value of its priority function would go down because of the increase of $h(\bar{p}_i, \text{a1})$, $h(\bar{p}_i, \text{a2})$ and $h(\bar{p}_i, \text{a3})$. Of course $\bar{p}_4$ is chosen as the second pattern. If $\alpha\times|\bar{p}_j|^2$ is not part of the priority function, both $f(\bar{p}_2)$ and $f(\bar{p}_4)$ will be 4, i.e., there is no preference to make a choice among these two. A random one will be taken. However, we can easily see that $\bar{p}_4$ is better than $\bar{p}_2$ in that $\bar{p}_4$ allows "b4" and "b5" to run in parallel.

Now a problem arises: How about $P_{def} = 1$ in the above example? That means only one pattern is allowed. Of course we have to use the pattern $\bar{p} = \{\text{ab}\}$ to be able to include all colors. Unfortunately there is no antichain with color set {a, b}, therefore pattern {ab} is not even a candidate! To solve this problem, the column number condition is used in the priority function, which will be explained below. The priority function is modified as follows:

$$f(\bar{p}_j) = \begin{cases} \displaystyle\sum_{n\in\mathcal{N}} \frac{h(\bar{p}_j, n)}{\displaystyle\sum_{\bar{p}_i\in\mathbb{P}_s} h(\bar{p}_i, n)+\varepsilon} + \alpha\times|\bar{p}_j|^2 & \text{if } \bar{p} \text{ satisfies the color number condition;} \\ 0 & \text{otherwise.} \end{cases}$$

Let the complete color set $\mathcal{L}$ represent all the colors that appear in the DFG,

$$\mathcal{L} = \{l(n)| \text{ for all } n \text{ in DFG}\}$$

and let the selected color set $\mathcal{L}_s$ represent all the colors that appear in one of already selected patterns, i.e.,

$$\mathcal{L}_s = \{l | l \in \bar{p}_j \text{ for } \bar{p}_j \in \mathbb{P}_s\}.$$

The new color set $\mathcal{L}_n(\bar{p})$ of the candidate pattern $\bar{p}$ consists of the colors that exist in $\bar{p}$ but not in the selected color set $\mathcal{L}_s$, i.e.,

$$\mathcal{L}_n(\bar{p}) = \{l | l \in \bar{p} \text{ and } l \notin \mathcal{L}_s\}.$$

We say that a candidate pattern satisfies the color pattern condition if the inequality (9) holds.

$$|\mathcal{L}_n(\bar{p})| \geq |\mathcal{L}| - |\mathcal{L}_s| - C\times(P_{def} - |\mathbb{P}_s| - 1). \quad (9)$$

$|\mathcal{L}| - |\mathcal{L}_s|$ is the number of colors that have not been covered by the $|\mathcal{L}_s|$ patterns. Except for the pattern that is going to be selected, there are another $(P_{def} - |\mathbb{P}_s| - 1)$ to be selected later, which can cover at most $C\times(P_{def} - |\mathbb{P}_s| - 1)$ uncovered different colors. Therefore the right part of the inequality is the minimum number of new colors that should be covered by the candidate pattern.

If we use a candidate pattern $\bar{p}$ which does not satisfy the inequality (9), some colors will not appear in the final chosen P_{def} patterns. For example, after selecting $(P_{def} - 1)$ patterns, there are still $(C + 2)$ colors that have never appeared in the selected $(P_{def} - 1)$ patterns. We can put at most C colors in the last pattern. Therefore the last two colors cannot appear in the patterns. To avoid this, when the inequality (9) is not satisfied for pattern $\bar{p}$, we do not select $\bar{p}$ by setting its priority function $f(\bar{p})$ to zero. If the priority function for all candidate patterns are zero, we have to make a pattern using C colors that have not appeared in the selected color set $\mathcal{L}_s$. The selection algorithm is modified and shown in Fig. 7.

Now let us do the example given in Fig. 4 again, assuming that only one pattern is allowed. In the inequality (9), $\mathcal{L} = \{\text{a}, \text{b}\}$, $\mathcal{L}_s = \Phi$, $P_{def} = 1$ and $\mathbb{P}_s = \phi$. The right side of the inequality is therefore 2. All the patterns generated from the graph have only one color. The new color sets for the four patterns are: $\mathcal{L}_n(\bar{p}_1) = \{a\}$, $\mathcal{L}_n(\bar{p}_2) = \{b\}$, $\mathcal{L}_n(\bar{p}_3) = \{a\}$, $\mathcal{L}_n(\bar{p}_4) = \{b\}$. Thus, $|\mathcal{L}_n(\bar{p}_1)| = |\mathcal{L}_n(\bar{p}_2)| = |\mathcal{L}_n(\bar{p}_3)| = |\mathcal{L}_n(\bar{p}_4)| = 1$. The inequality does not hold for any of them. Due to the presented modification a new pattern {ab} is made.

```
1   for(i = 0; i < P_def; i++) {
2       Compute the priority function for each pattern.
3       Choose the pattern with the largest nonzero
        priority function. If there is no pattern with
        nonzero priority function, take C uncovered col-
        ors to make a pattern.
4       Delete the subpatterns of the selected pattern.
5   }
```

Figure 7. The pseudo-code for modified pattern selection procedure

Table 7. Experimental result of the pattern selection algorithm.

	3DFT		5DFT	
P_{def}	Random	Selected	Random	Selected
1	12.4	8	23.4	19
2	10.5	7	22	16
3	8.7	7	20.4	16
4	7.9	7	15.8	15
5	6.5	6	15.8	15

6 Experiment

We ran the multi-pattern scheduling algorithm on the 3- and 5- Fast Fourier Transform (3DFT and 5DFT) algorithms. The experimental results are given in Table 7, where the number indicates the number of clock cycles needed. The data in the columns "Random" are computed by using the randomly generated patterns, while the columns "Selected" are computed using the patterns selected by the presented algorithm. Random patterns are tested ten times and the average of the results is put into the table. From the simulation results we have the following observations:

1. As more patterns are allowed the number of needed clock cycles gets smaller. This is the benefit achieved by using reconfiguration.

2. The patterns selected by the presented algorithm lead to better scheduling result than randomly generated patterns.

7 Conclusions

This paper presents an algorithm to select a set of patterns for a multi-pattern scheduling algorithm, which is designed to schedule a graph to a coarse-grained reconfigurable architecture – Montium. An heuristic approach is adopted in the algorithm, which chooses the most frequently appearing patterns by using a priority function. The experiments show that the patterns selected by the algorithm will lead to better scheduling results. The proposed approach makes the further improvement very simple: by just modifying the priority function. In our future work we will go on working on the priority function to improve the performance.

References

[1] Yuanqing Guo, Cornelis Hoede, and Gerard J.M. Smit, "A Multi-Pattern Scheduling Algorithm" to appear in the *Final Edition of the proceeding of ERSA 2005*, June 27-30, 2005, Monte Carlo Resort, Las Vegas, Nevada, USA.

[2] Paul M. Heysters, Gerard J.M. Smit, E. Molenkamp: "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems", *The Journal of Supercomputing*, Vol 26, No. 3, Kluwer Academic Publishers, Boston, U.S.A., November 2003, ISSN 0920-8542.

[3] Yuanqing Guo, Gerard J.M. Smit, Hajo Broersma, Michel A.J. Rosien, Paul M. Heysters, "Mapping Applications to a Coarse Grain Reconfigurable System", In *Proceedings of 8th Asia-Pacific Conference (ACSAC 2003)*, Aizu-Wakamatsu, Japan, September 23-26, 2003, 221-235.

[4] Robert A. Walker and Samit Chaudhuri, "High-Level Synthesis: Introduction to the Scheduling Problem", *IEEE Design and Test* 12(2):60-69, Summer 1995.

[5] D. Bernstein, M. Rodeh, and I. Gertner, "On the Complexity of Scheduling Problems for Parallel/Pipelined Machines", *IEEE Transactions on Computers*, 38(9):130813, September 1989.

[6] B.M. Pangrle and D.D. Gajski, "Design Tools for Intellegent Compilation," *IEEE Trans. Computer-Aided Design,* Vol. CAD-6, No. 6, Nov.1987, pp. 1098-1112.

[7] T.C. Hu, "Parallel Sequencing and Assembly Line Problems," *Operations Research,* Vol.9, No.6, Nov.1961. pp. 841-848.

[8] P.G. Paulin and J.P. Knight, "Algorithms for High-Level Synthesis," *IEEE Design and Test of Computers,* Vol.6, No.4, Dec. 1989, pp.18-31.

[9] Eric W. Weisstein. "Antichain." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Antichain.html