Improved Force-Directed Scheduling

W.F.J. Verhaegh[1] E.H.L. Aarts[1,2] J.H.M. Korst[1] P.E.R. Lippens[1]

[1] Philips Research Laboratories
P.O. Box 80000, 5600 JA Eindhoven
The Netherlands

[2] Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven
The Netherlands

Abstract

We present a mathematical justification of the technique of force-directed scheduling and propose two modifications of the basic algorithm introduced by Paulin and Knight. The newly presented modifications improve the effectiveness of force-directed scheduling without affecting its time complexity. This is illustrated by an empirical performance analysis based on a number of problem instances.

1 Introduction

An important problem in high-level synthesis is that of scheduling operations such that resources and/or the total execution time are minimized. Most scheduling problems have such inherent complexity that it is generally assumed that any algorithm that solves them to optimality requires a computational effort that grows superpolynomially with the size of the problem. Therefore, several approximation algorithms have been developed, which have a polynomial time complexity, but cannot guarantee to find optimal solutions; see for instance [6, Section 3.1] and [8, Section III].

Recently, Paulin and Knight [7,8] introduced a new approximation algorithm based on the technique of force-directed scheduling. The intent of force-directed scheduling is to minimize the number of resources by smoothing the distribution of the operations in time without increasing the total execution time or makespan. Since its introduction, the technique of force-directed scheduling has gained interest from other research groups [2,9,10]. In [9] a simplification of force-directed scheduling is used, and in [2], force-directed scheduling is combined with hardware assignment.

Our studies of force-directed scheduling, in order to use it in the PHIDEO silicon compiler [5], have been concentrating on a mathematical formulation of the force-directed scheduling technique that justifies the original approach proposed by Paulin and Knight. Furthermore, we have been investigating potential modifications that may improve the effectiveness of the algorithm without changing its time complexity. Two such modifications are:

- gradual time-frame reduction, and
- the use of global spring constants.

In this paper we report on some of the results obtained in our study. First we present a mathematical model from which the basic force-directed scheduling algorithm proposed by Paulin and Knight can be derived, in order to get more insight in how the algorithm works, in Section 2. In Section 3 we describe the modifications and we show that they do not change the time complexity. Furthermore, we argue that the modifications will improve the algorithm. Finally, in Section 4, we present some experimental results of the modified algorithm and compare them to results obtained with the original algorithm.

2 Basic Force-Directed Scheduling

In this section we present a mathematical justification of the basic force-directed scheduling algorithm introduced by Paulin and Knight. A mathematical model is introduced to give a better understanding of the basic concepts of force-directed scheduling.

2.1 Problem Definition

We consider the following scheduling problem.

Definition 1 *[Scheduling to Minimize Resources]*
Given is a set $\mathcal{R}$ of resource types, a cost $w_r > 0$ for each $r \in \mathcal{R}$, a set $\mathcal{O} = \{1,2,\ldots,n\}$ of unit-time operations, a resource type r_i for each $i \in \mathcal{O}$, a precedence relation $\prec$ on $\mathcal{O}$, and a makespan m. A schedule $\tau = (t_1,\ldots,t_n)$ assigns to each operation $i \in \mathcal{O}$ a time $t_i \in \mathcal{T} = \{1,2,\ldots,m\}$, at which i is executed. A schedule τ is called feasible if and only if

$$\forall_{i,j\in\mathcal{O}} : i \prec j \Rightarrow t_i < t_j.$$

Now the problem is to find a feasible schedule τ that minimizes the resource costs, given by

$$f(\tau) = \sum_{r\in\mathcal{R}} w_r \max_{t\in\mathcal{T}} N_r(\tau,t),$$

where $N_r(\tau,t) = |\{i \in \mathcal{O} \mid r_i = r \wedge t_i = t\}|$ is the number of operations of type r scheduled at time t in schedule τ. This $N_r(\tau,t)$ is called a requirement function.

The decision variant of this scheduling problem is NP-complete. A reduction from Precedence Constrained Scheduling [SS9 in 3] is straightforward.

To obtain a better understanding of the force-directed scheduling algorithm we (i) reformulate the problem by giving an approximating cost function and (ii) indicate how an iterative approach can be used to find good solutions for this alternative problem formulation.

2.2 Approximating Cost Function

The cost function $f(\tau)$ can be rewritten as

$$f(\tau) = \sum_{r \in \mathcal{R}} w_r \mu_r + \sum_{r \in \mathcal{R}} w_r \max_{t \in \mathcal{T}} (N_r(\tau, t) - \mu_r),$$

where $\mu_r = \frac{1}{m} |\{i \in \mathcal{O} | r_i = r\}|$ is the average number of operations of type r over $\mathcal{T}$. The first term in $f(\tau)$ can be left out since it is a constant, and for reasons of convenience we approximate the remaining term by $f'(\tau)$, given by

$$f'(\tau) = \sum_{r \in \mathcal{R}} w_r \sum_{t \in \mathcal{T}} (N_r(\tau, t) - \mu_r)^2.$$

This approximating cost function reflects the aim of minimizing the resource costs by smoothing the distribution of resources in time, i.e. by minimizing the deviation from the average value. The choice of an alternative approximating cost function f' will be the subject of future research; see also Section 3.2.

2.3 Iterative Approach

Next we discuss how an iterative constructive algorithm can be used to find solutions for which f' is near-optimal. The algorithm constructs a solution through a sequence of partial solutions, where in each iteration an operation is assigned to a time. The set of partial solutions can be defined as follows. Instead of choosing one time t_i for each operation i, we choose a *time frame* $\{a_i, ..., b_i\}$, $a_i, b_i \in \mathcal{T}$, $a_i \leq b_i$, which means that operation i has to be scheduled at a time $t_i \in \{a_i, ..., b_i\}$. In this way we get a *schedule frame* $\tilde{\tau} = (a_1, b_1, ..., a_n, b_n)$, which we call feasible if and only if

$$\forall_{i,j \in \mathcal{O}} : i \prec j \Rightarrow a_i < a_j \ \wedge \ b_i < b_j.$$

Now the set of partial solutions is given by the set of feasible schedule frames. Note that basically an operation i can be scheduled at any time t_i in its time frame. Furthermore, note that a (feasible) schedule frame $\tilde{\tau}$ with $a_i = b_i$ for all $i \in \mathcal{O}$ corresponds to a (feasible) schedule τ.

Next, the costs of a partial solution are estimated as follows. We define *requirement distribution functions* $\tilde{N}_r(\tilde{\tau}, t)$ as the estimated number of operations of type r at time t in schedule frame $\tilde{\tau}$. For this, we determine the probability $P(\tilde{\tau}, i, t)$ that operation i will eventually be scheduled at time t, considering schedule frame $\tilde{\tau}$. Because of the precedence relation, these probabilities are generally not independent. Nevertheless, we estimate these probabilities by assuming uniform probability of assigning an operation to any time in its time frame, i.e.

$$P(\tilde{\tau}, i, t) = \begin{cases} \frac{1}{b_i - a_i + 1} & \text{if } a_i \leq t \leq b_i, \\ 0 & \text{otherwise.} \end{cases}$$

Now for each resource type r we take the summation of probabilities of the operations for each time $t \in \mathcal{T}$. The resulting requirement distribution functions are given by

$$\tilde{N}_r(\tilde{\tau}, t) = \sum_{i \in \mathcal{O}_r} P(\tilde{\tau}, i, t),$$

where $\mathcal{O}_r = \{i \in \mathcal{O} | r_i = r\}$. Next we define a cost function for schedule frames (comparable with the approximating cost function for schedules) by

$$\tilde{f}(\tilde{\tau}) = \sum_{r \in \mathcal{R}} w_r \sum_{t \in \mathcal{T}} (\tilde{N}_r(\tilde{\tau}, t) - \mu_r)^2.$$

Note that if $a_i = b_i = t_i$, for all $i \in \mathcal{O}$, then $\tilde{N}_r(\tilde{\tau}, t) = N_r(\tau, t)$, and thus $\tilde{f}(\tilde{\tau}) = f'(\tau)$. So we want to find a schedule frame $\tilde{\tau}$ which minimizes $\tilde{f}(\tilde{\tau})$, under the constraint $a_i = b_i$ for all $i \in \mathcal{O}$.

The iterative approach is now as follows. First the schedule frame $\tilde{\tau}$ is initialized by initializing the a_i's with the times of the ASAP (as soon as possible) schedule and the b_i's with the times of the ALAP (as late as possible) schedule. Note that from this partial solution each feasible schedule can be obtained. Next in each iteration the *neighbourhood space* $S(\tilde{\tau})$ is explored, which is the set of feasible schedule frames $\tilde{\sigma}$ obtained from $\tilde{\tau}$ by scheduling an operation i with $a_i < b_i$ at a time $t \in \{a_i, ..., b_i\}$ and updating the time frames of the other operations. The assignment of an operation to a time is chosen for which

$$\Delta \tilde{f}(\tilde{\tau}, \tilde{\sigma}) = \tilde{f}(\tilde{\sigma}) - \tilde{f}(\tilde{\tau})$$

is minimal. Iterations are repeated until a feasible schedule is obtained. This iterative approach is identical to basic force-directed scheduling as defined by Paulin and Knight except that there the assignment of operation i to time t is chosen for which a *total force* $F_{tot}(i, t)$ is minimal, instead of $\Delta \tilde{f}$. However, if we define

$$\Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s) = \tilde{N}_r(\tilde{\sigma}, s) - \tilde{N}_r(\tilde{\tau}, s),$$

for all $s \in \mathcal{T}$ and $r \in \mathcal{R}$, then

$$\begin{aligned}
&\Delta \tilde{f}(\tilde{\tau}, \tilde{\sigma}) \\
&= \sum_{r \in \mathcal{R}} w_r \sum_{s \in \mathcal{T}} [(\tilde{N}_r(\tilde{\sigma}, s) - \mu_r)^2 - (\tilde{N}_r(\tilde{\tau}, s) - \mu_r)^2] \\
&= \sum_{r \in \mathcal{R}} w_r \sum_{s \in \mathcal{T}} [(\tilde{N}_r(\tilde{\tau}, s) + \Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s) - \mu_r)^2 \\
&\qquad\qquad - (\tilde{N}_r(\tilde{\tau}, s) - \mu_r)^2] \\
&= \sum_{r \in \mathcal{R}} w_r \sum_{s \in \mathcal{T}} [(\Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s))^2 + 2\tilde{N}_r(\tilde{\tau}, s) \Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s) \\
&\qquad\qquad - 2\Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s) \mu_r] \\
&= 2 \sum_{r \in \mathcal{R}} w_r \sum_{s \in \mathcal{T}} (\tilde{N}_r(\tilde{\tau}, s) + \tfrac{1}{2} \Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s)) \Delta \tilde{N}_r(\tilde{\tau}, \tilde{\sigma}, s),
\end{aligned}$$

since $\sum_{s\in T} \Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s) = 0$. Hence,

$$\frac{1}{2}\Delta\tilde{f}(\tilde{\tau},\tilde{\sigma}) = \sum_{r\in R} w_r \sum_{s\in T} \tilde{N}_r(\tilde{\tau},s)\Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s) + \sum_{r\in R} w_r \sum_{s\in T} \tfrac{1}{2}(\Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s))^2. \quad (1)$$

As we will show in the following sections, the first term in this equation is equal to F_{tot} and the second term strongly resembles the look-ahead introduced by Paulin and Knight.

2.4 Forces

For the selection of the assignment of an operation i to a time t Paulin and Knight introduced a total force $F_{tot}(i,t)$ as follows. First the changes in the probabilities $P(\tilde{\tau},j,s)$ are determined, given by

$$\Delta p(\tilde{\tau},\tilde{\sigma},j,s) = P(\tilde{\sigma},j,s) - P(\tilde{\tau},j,s),$$

where $\tilde{\sigma}$ is the feasible schedule frame obtained from $\tilde{\tau}$ by assigning $a_i = b_i = t$ and updating the time frames of the other operations. Then a ***self force*** $F_{self}(i,t)$, a ***successor force*** $F_{succ}(i,t)$, and a ***predecessor force*** $F_{pred}(i,t)$ are determined, given by

$$\begin{aligned} F_{self}(i,t) &= \sum_{s\in T} \Delta p(\tilde{\tau},\tilde{\sigma},i,s)\tilde{N}_{r_i}(\tilde{\tau},s)w_{r_i}, \\ F_{succ}(i,t) &= \sum_{j\in Succ(i)} \sum_{s\in T} \Delta p(\tilde{\tau},\tilde{\sigma},j,s)\tilde{N}_{r_j}(\tilde{\tau},s)w_{r_j}, \\ F_{pred}(i,t) &= \sum_{j\in Pred(i)} \sum_{s\in T} \Delta p(\tilde{\tau},\tilde{\sigma},j,s)\tilde{N}_{r_j}(\tilde{\tau},s)w_{r_j}, \end{aligned}$$

where $Succ(i)$ is the set of successor operations of operation i, and $Pred(i)$ is the set of predecessor operations of operation i. These forces can be seen as forces needed for displacements $\Delta p(\tilde{\tau},\tilde{\sigma},j,s)$ of springs with spring constants $\tilde{N}_{r_j}(\tilde{\tau},s)w_{r_j}$. Now $F_{tot}(i,t)$ is given by

$$\begin{aligned} F_{tot}(i,t) &= F_{self}(i,t) + F_{succ}(i,t) + F_{pred}(i,t) \\ &= \sum_{j\in O} \sum_{s\in T} \Delta p(\tilde{\tau},\tilde{\sigma},j,s)\tilde{N}_{r_j}(\tilde{\tau},s)w_{r_j} \\ &= \sum_{r\in R} w_r \sum_{s\in T} \sum_{j\in O_r} \Delta p(\tilde{\tau},\tilde{\sigma},j,s)\tilde{N}_r(\tilde{\tau},s) \\ &= \sum_{r\in R} w_r \sum_{s\in T} \Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s)\tilde{N}_r(\tilde{\tau},s), \end{aligned}$$

which is equal to the first term in Equation 1.

2.5 Look-Ahead

To improve the effectiveness of the force-directed scheduling algorithm, Paulin and Knight [8] presented a simple look-ahead scheme. The idea is to replace the factor $\tilde{N}_{r_j}(\tilde{\tau},s)$ in the force calculations by a value somewhere between the current value and the value that would be obtained after the current iteration. Paulin and Knight propose to calculate the forces as sums of terms given by

$$\Delta p(\tilde{\tau},\tilde{\sigma},j,s)(\tilde{N}_{r_j}(\tilde{\tau},s) + \eta\Delta p(\tilde{\tau},\tilde{\sigma},j,s))w_{r_j},$$

in which Paulin and Knight choose $\eta = \frac{1}{3}$.

However, it is better to replace $\tilde{N}_{r_j}(\tilde{\tau},s)$ by $\tilde{N}_{r_j}(\tilde{\tau},s) + \eta\Delta\tilde{N}_{r_j}(\tilde{\tau},\tilde{\sigma},s)$, since the value after the current iteration is equal to $\tilde{N}_{r_j}(\tilde{\tau},s) + \Delta\tilde{N}_{r_j}(\tilde{\tau},\tilde{\sigma},s)$. Then the forces are sums of terms given by

$$\Delta p(\tilde{\tau},\tilde{\sigma},j,s)(\tilde{N}_{r_j}(\tilde{\tau},s) + \eta\Delta\tilde{N}_{r_j}(\tilde{\tau},\tilde{\sigma},s))w_{r_j}.$$

The additional term to the total force is then given by

$$\begin{aligned} &\sum_{j\in O} \sum_{s\in T} \eta\Delta p(\tilde{\tau},\tilde{\sigma},j,s)\Delta\tilde{N}_{r_j}(\tilde{\tau},\tilde{\sigma},s)w_{r_j} \\ = &\sum_{r\in R} w_r \sum_{s\in T} \eta \sum_{j\in O_r} \Delta p(\tilde{\tau},\tilde{\sigma},j,s)\Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s) \\ = &\sum_{r\in R} w_r \sum_{s\in T} \eta(\Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s))^2. \end{aligned}$$

If $\eta = \frac{1}{2}$, then this additional term is exactly equal to the second term in Equation 1, so then the $\Delta\tilde{f}$ criterion and the force criterion are equivalent.

However, experimental results have shown that $\eta = \frac{1}{3}$ is a better value, so in the following sections we use the criterion $\Delta C(\tilde{\tau},\tilde{\sigma})$, which is equal to

$$\sum_{r\in R} w_r \sum_{s\in T} (\tilde{N}_r(\tilde{\tau},s) + \eta\Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s))\Delta\tilde{N}_r(\tilde{\tau},\tilde{\sigma},s),$$

in which $\eta = \frac{1}{3}$.

2.6 The Algorithm

The basic force-directed scheduling algorithm can be written as follows.

Step 1 Initialize time frames with ASAP and ALAP schedule.
Step 2 Calculate requirement distribution functions.
Step 3 For each operation i which is not scheduled yet, and each time $t \in \{a_i,\ldots,b_i\}$: calculate $\Delta C(\tilde{\tau},\tilde{\sigma})$, where $\tilde{\sigma}$ is the feasible schedule frame obtained from $\tilde{\tau}$ by scheduling operation i at time t.
Step 4 Schedule that operation i at that time t for which $\Delta C(\tilde{\tau},\tilde{\sigma})$ is minimal; i.e. assign $a_i = b_i = t$.
Step 5 Update time frames of predecessors and successors of i.
Step 6 Update requirement distribution functions.
Step 7 If not all operations are scheduled: goto Step 3.

Here we say that an operation i is scheduled if and only if $a_i = b_i$.

2.7 An Example

For an example with 1 resource type, with cost 1, 9 operations, and a makespan of 5, we determined the ASAP and

ALAP schedule; see Figure 1. The precedence relations are denoted by arrows. For this example the time frames are initiated $(a_1, b_1) = (1,2)$, $(a_2, b_2) = (1,2)$, $(a_3, b_3) = (2,3)$, etc.

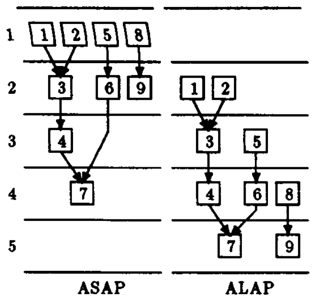

Figure 1: *An ASAP and an ALAP schedule*

The values of the requirement distribution function for this initial schedule frame $\tilde{\tau}$ are $\tilde{N}_1(\tilde{\tau},1) = \frac{1}{2} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{19}{12}$, $\tilde{N}_1(\tilde{\tau},2) = \frac{8}{3}$, $\tilde{N}_1(\tilde{\tau},3) = \frac{13}{6}$, $\tilde{N}_1(\tilde{\tau},4) = \frac{11}{6}$, and $\tilde{N}_1(\tilde{\tau},5) = \frac{3}{4}$. For scheduling operation 8 at time 3 the changes in the distribution function are $\Delta\tilde{N}_1(\tilde{\tau},\tilde{\sigma},1) = -\frac{1}{4}$, $\Delta\tilde{N}_1(\tilde{\tau},\tilde{\sigma},2) = -\frac{1}{2}$, $\Delta\tilde{N}_1(\tilde{\tau},\tilde{\sigma},3) = \frac{1}{2}$, $\Delta\tilde{N}_1(\tilde{\tau},\tilde{\sigma},4) = 0$, and $\Delta\tilde{N}_1(\tilde{\tau},\tilde{\sigma},5) = \frac{1}{4}$. So $\Delta C(\tilde{\tau},\tilde{\sigma}) = -\frac{1}{4}(\frac{19}{12} - \frac{1}{12}) - \frac{1}{2}(\frac{8}{3} - \frac{1}{6}) + \frac{1}{2}(\frac{13}{6} + \frac{1}{6}) + 0(\frac{11}{6} + 0) + \frac{1}{4}(\frac{3}{4} + \frac{1}{12}) = -\frac{1}{4}$.

2.8 Time Complexity

The algorithm described above requires $O(m^2n^3)$ steps when implemented in a straightforward way, where m is the makespan and n is the number of operations. This complexity can be derived as follows.

- In each iteration at least one operation is scheduled (scheduling an operation may cause successor or predecessor operations to be scheduled implicitly), so at most n iterations are needed.
- In each iteration, there are at most n operations that still must be scheduled.
- For each of these operations there are at most m times to which it can be assigned.
- For each tentative assignment of an operation to a time, the probabilities of $O(n)$ operations may change and there are m times where they may change, so calculating ΔC takes $O(mn)$ steps.

This worst-case time complexity is identical to the time complexity of a straightforward implementation of the algorithm proposed by Paulin and Knight, although they do not mention the second factor m which arises from the summation of products over m times for each force. Furthermore, Paulin and Knight claim that the complexity can be reduced to $O(m^2n^2)$ when using an alternative method for computing the forces [8]. However, this alternative method does not yield the original forces; it yields a different set of forces, due to the fact that the contribution of a successor/predecessor force to $F_{tot}(i,t)$ is multiply counted.

3 Modifications

In this section we discuss two modifications of the basic algorithm: gradual time-frame reduction, and the use of global spring constants.

3.1 Gradual Time-Frame Reduction

Instead of reducing in each iteration the time frame $\{a_i, \ldots, b_i\}$ of an operation i to one time t, which is quite a large step, we decide to reduce the time frame with only one clock cycle, i.e. to $\{a_i + 1, \ldots, b_i\}$ or $\{a_i, \ldots, b_i - 1\}$. We do this as follows.

Firstly we compute for each operation i for which $a_i < b_i$ the differences ΔC of scheduling operation i at time a_i and at time b_i, which we denote by $d_{left}(i)$ and $d_{right}(i)$, respectively. Next we determine $d_{min}(i) = \min\{d_{left}(i), d_{right}(i)\}$, and $d_{max}(i) = \max\{d_{left}(i), d_{right}(i)\}$. Then we determine

$$d^-_{min}(i) = \begin{cases} d_{min}(i) & \text{if } a_i + 1 = b_i, \\ \min\{d_{min}(i), 0\} & \text{if } a_i + 1 < b_i, \end{cases}$$

which is a rough estimate for the minimum ΔC of scheduling operation i at a time $t \in \{a_i, \ldots, b_i\}$. Now $d_{max}(i)$ reflects the worst possible effect of scheduling operation i at time a_i or b_i, and $d^-_{min}(i)$ reflects the best possible effect of scheduling operation i somewhere in its time frame. Next we determine $\Delta d_{gain}(i) = d_{max}(i) - d^-_{min}(i) \geq 0$, which reflects the attainable improvement by reducing the time frame of operation i.

Now we choose operation i for which $\Delta d_{gain}(i)$ is maximal and we reduce its time frame to $\{a_i + 1, \ldots, b_i\}$ if $d_{left}(i) \geq d_{right}(i)$, and to $\{a_i, \ldots, b_i - 1\}$ if $d_{left}(i) < d_{right}(i)$.

Then time frames and distribution functions are updated, and the same procedure is repeated until $a_i = b_i$ for all $i \in O$, which corresponds to a schedule.

Now the time frames of the operations are gradually shrunk, and the distribution functions become gradually better estimates of the final distributions of the operations. In this way the eventual assignment of operations to times is postponed to moments when the distribution functions are better estimates. This modification makes the algorithm less greedy, and we may expect that the algorithm generally obtains better solutions.

The time complexity of the algorithm is $O(m^2n^3)$, which is the same as that of the basic force-directed scheduling algorithm. This can be seen as follows. In each iteration at least one time is subtracted from a time frame of an operation, so there are at most mn iterations needed. On the other hand, in each iteration a ΔC for only two tentative assignments has to be calculated for each operation, instead of $O(m)$ ones.

3.2 Global Spring Constants

The second modification we propose is the use of global spring constants. To explain this, we use the force model of the algorithm. In the force calculations spring constants $\tilde{N}_r(\bar{\tau},s)w_r$ are used, and due to the factor w_r trade-offs are performed between the different types of resources. Since we want to minimize

$$\sum_{r\in R} w_r M_r(\bar{\tau}),$$

where $M_r(\bar{\tau}) = \max_{t\in T} \tilde{N}_r(\bar{\tau},t)$, it is not desirable to decrease $\tilde{N}_{r_1}(\bar{\tau},t_1)$ at the expense of increasing $\tilde{N}_{r_2}(\bar{\tau},t_2)$ if $\tilde{N}_{r_1}(\bar{\tau},t_1) \ll M_{r_1}(\bar{\tau})$ and $\tilde{N}_{r_2}(\bar{\tau},t_2) \approx M_{r_2}(\bar{\tau})$. However, negative contributions to F_{tot} of resource type r_1 might compensate positive contributions to F_{tot} of type r_2. Therefore, we suggest computing forces with spring constants inversely proportional to the difference between a distribution function and its maximum:

$$w_r/(M_r(\bar{\tau}) - \tilde{N}_r(\bar{\tau},s) + \varepsilon) \quad \text{instead of} \quad \tilde{N}_r(\bar{\tau},s)w_r,$$

where ε is a small positive constant to prevent divisions by zero and to prevent that the maxima of the distribution functions are too much weighted, in comparison to near-maximal values. In this way changes in probabilities are more important at times and for resource types with (near-) maximal values of the distribution functions, and performing trade-offs is done more sensibly. Note that this change in spring constants in fact means that we change the approximating cost function.

The extra work to be done is the determination of the maxima of the distribution functions in each iteration. This is $\mathcal{O}(mn)$ in each iteration, but the rest of the steps in an iteration is $\mathcal{O}(mn^2)$, so the time complexity of the modified algorithm is still $\mathcal{O}(m^2n^3)$.

For the look-ahead we again replace $\tilde{N}_{r_j}(\bar{\tau},s)$ by $\tilde{N}_{r_j}(\bar{\tau},s) + \eta\Delta\tilde{N}_{r_j}(\bar{\tau},\bar{\sigma},s)$. The forces now are sums of terms given by

$$\frac{\Delta p(\bar{\tau},\bar{\sigma},j,s)w_{r_j}}{\varepsilon + (M_{r_j}(\bar{\tau}) - \tilde{N}_{r_j}(\bar{\tau},s) - \eta\Delta\tilde{N}_{r_j}(\bar{\tau},\bar{\sigma},s))^+}$$

in which $x^+ = \max\{x,0\}$, to prevent negative values. Experimental results have shown that a good value for ε is $\varepsilon = \frac{1}{5}$.

4 Experimental Results

For the force-directed scheduling algorithm Paulin and Knight [8] introduced a set of refinements to include multi-time operations, interconnect costs, etc. These refinements can also be included in the modified algorithm.

4.1 Fifth-Order Digital Elliptical Wave Filter

This example is taken from Kung, Whitehouse, and Kailath [4] and it was chosen as a benchmark for the 1988 High-Level Synthesis Workshop [1]. For this example, we minimize the number of functional units, by taking $w = 1$ for both the adders and the multipliers. In this example adders take one time unit to execute and multipliers take two time units. We apply four algorithms: the basic algorithm given in Section 2.6 (FDS), the algorithm with gradual time-frame reduction (GTFR), the algorithm with global spring constants (GSC), and the algorithm with both modifications (MFDS). For all algorithms we take $\eta = \frac{1}{3}$, and for GSC and MFDS we take $\varepsilon = \frac{1}{5}$. In Table 1 we summarize the functional unit allocation for different makespans obtained by the different algorithms. The total costs of these results are the number of adders plus the number of multipliers. CPU times vary between two seconds and three minutes on an Apollo DN10000.

Table 1: *Wave filter FU allocations*

	makespan			
algorithm	17	18	19	21
FDS	$+++$	$+++$	$++$	$++$
	$\times\times\times$	$\times\times$	$\times\times$	$\times$
GTFR	$+++$	$+++$	$++$	$++$
	$\times\times\times$	$\times\times$	$\times\times$	$\times$
GSC	$+++$	$++$	$++$	$++$
	$\times\times\times$	$\times\times$	$\times\times$	$\times\times$
MFDS	$+++$	$++$	$++$	$++$
	$\times\times\times$	$\times\times$	$\times\times$	$\times$
FDS,GTFR,	$+++$	$+++$	$++$	
GSC, MFDS	$\otimes\otimes$	$\otimes$	$\otimes$	

$+$ adder, $\times$ multiplier, $\otimes$ pipelined multiplier

All algorithms give optimal results for all cases, except FDS and GTFR for a makespan of 18 and regular multipliers, and GSC for a makespan of 21 and regular multipliers.

As we can see, in the case of a makespan of 18 and pipelined multipliers, there are two solutions with 4 functional units: 3 adders + 1 multiplier, or 2 adders + 2 multipliers (a pipelined multiplier can do at least the same as a regular multiplier). If the multiplier costs are higher than the adder costs, then the first solution is optimal, otherwise the second solution is optimal. We executed the algorithms for different costs and the results are shown in Table 2.

Table 2: *FU allocation for different costs, makespan is 18*

algorithm	$w_\otimes = 2w_+$	$w_+ = 2w_\otimes$
FDS, GTFR	$+++$	$+++$
	$\otimes$	$\otimes$
GSC, MFDS	$+++$	$++$
	$\otimes$	$\otimes\otimes$

As we can see, the algorithms with global spring constants find optimal solutions for both cases, while the other two do not find an optimal solution if $w_+ = 2w_\otimes$.

4.2 A More Comprehensive Example

A second example we used is more comprehensive. We have a data flow graph consisting of four relatively independent subgraphs which overlap each other in time. In this graph we have different types of functional units with different costs and execution times. Furthermore, we take the number of variables alive at the same time as an estimate for the number of registers and we have an estimate for interconnect, each with a separate cost factor. Further details of this example can be obtained from the authors.

The results for this example are shown in Table 3.

Table 3: *Results of the more comprehensive example*

algorithm	total costs
FDS	44.60
GTFR	39.60
GSC	30.85
MFDS	29.55

For this example 29.55 is the best result we ever found, and we see that FDS produces a solution for which the costs are over 50% higher than this value. We can also see that the largest improvement is due to the use of global spring constants.

5 Conclusions

We have given a mathematical justification of the basic force-directed scheduling algorithm as introduced by Paulin and Knight [7,8]. Furthermore, we proposed two modifications of the force-directed scheduling algorithm: gradual time-frame reduction, which makes the algorithm less greedy, and global spring constants, which ensures that trade-offs are done better. These modifications improve the effectiveness of the algorithm without increasing its time complexity.

References

[1] G. Borriello, E. Detjens, High-Level Synthesis: Current Status and Future Directions, *Proceedings of the 25th Design Automation Conference*, Anaheim, CA, July 1988, pp. 477-482.

[2] R.J. Cloutier, D.E. Thomas, The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm, *Research Report No. CMUCAD-90-17*, Carnegie Mellon University, Pittsburgh, PA, May 1990.

[3] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, NY, 1979, page 239.

[4] S.Y. Kung, H.J. Whitehouse, T. Kailath, *VLSI and Modern Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1985, pp. 258-264.

[5] P.E.R. Lippens *et al.*, PHIDEO: A Silicon Compiler for High Speed Algorithms, to be published in *Proceedings European Design Automation Conference*, Amsterdam, the Netherlands, February 1991.

[6] M.C. McFarland, A.C. Parker, R. Camposano, Tutorial on High-Level Synthesis, *Proceedings of the 25th Design Automation Conference*, Anaheim, CA, July 1988, pp. 330-336.

[7] P.G. Paulin, J.P. Knight, Force-Directed Scheduling in Automatic Data Path Synthesis, *Proceedings of the 24th Design Automation Conference*, Miami Beach, FL, July 1987, pp. 195-202.

[8] P.G. Paulin, J.P. Knight, Force-Directed Scheduling for the Behavioural Synthesis of ASIC's, *IEEE Transactions on Computer Aided Design*, vol. 8, no. 6, June 1989, pp. 661-679.

[9] L. Stok, R. van den Born, EASY: Multiprocessor Architecture Optimization, *Proceedings International Workshop Logic and Architecture Synthesis for Silicon Compilers*, Grenoble, France, May 1988.

[10] W.F.J. Verhaegh, Scheduling Problems in Video Signal Processing, *Philips Technical Note NL-TN 089/90*, April 1990, pp. 56-70.