Adaptive streaming applications : analysis and implementation models

Zhai, J.T.

Chapter 5

Exploiting Just-enough Parallelism in Hard Real-time Systems

Jiali Teddy Zhai, Mohamed A. Bamakhrama, Todor Stefanov, "Exploiting Just-enough Parallelism when Mapping Streaming Applications in Hard Real-time Systems", *In the Proceedings of the 50th IEEE/ACM Design Automation Conference (DAC'13)*, pp. 170:1–170:8, Austin, TX, USA, June 2 - 6, 2013.

As we have seen in Chapter 4, the initial application specification, often in the form of a graph, may be transformed to an alternative one that exposes more parallelism while preserving the same application behavior. To this end, task unfolding is an effective technique to generate such alternative graphs. Basically, task unfolding replicates the functionality of a task by a certain number of times, referred as *unfolding factor*. Then, replicas of tasks concurrently process different data, thereby exploring also data-level parallelism next to the task-level parallelism. For data flow MoCs, such as SDF, the unfolding has been extensively applied in [40, 54, 56, 71].

In the context of Daedalus^{RT}, unfolding individual actors in an initial SDF graph by different unfolding factors results in a large number of possible alternative CSDF graphs. To transform the initial SDF graph to an alternative CSDF one by unfolding, the main problem is to determine a proper unfolding factor for each task. This problem is challenging because platform constraints must be considered during unfolding. The platform constraints can be the number of available PEs and temporal scheduling of actors on the PEs. In Chapter 4 and other literature [40, 133], an unfolding factor[1] is determined for each task in such a way that the obtained

[1]Our communication-free partitioning presented in Chapter 4 on PPN processes can be considered

alternative graph exposes the maximum DLP without considering the platform constraints. However, unfolding a task too many times reveals more parallelism than the processing capability of the execution platform. The overwhelming parallelism leads to an inefficient mapping of replicas of tasks. That is, the excessive number of replicas cannot be efficiently allocated and temporally scheduled on the available PEs. Moreover, the excessive number of replicas introduces significant memory overhead for both code and data. On the other hand in [54, 71, 113], the authors assume that the unfolding factor of a task cannot exceed the number of available PEs on the execution platform. This assumption, however, restricts the amount of revealed parallelism because a proper unfolding factor is not necessarily less than or equal to the number of available PEs. As a consequence, the aforementioned assumption might lead to under-utilized PEs. From the discussion above, we can see that exploiting excessive or insufficient parallelism may result in sub-optimal system utilization and performance. Therefore, in this chapter, we address the problem of determining a proper unfolding factor of each SDF actor in a given initial SDF graph, such that the obtained alternative CSDF graph exposes *just-enough* parallelism to fully utilize the available PEs. This is achieved by considering the platform constraints when determining the unfolding factors.

Scope of Work

In this chapter, we assume that a given SDF graph is acyclic. Note that this assumption is not directly related to our approach in this chapter. Rather, it is merely a restriction of the adopted hard real-time scheduling framework (see Section 2.3 on page 33). Such assumption covers a large set of applications as it has been empirically shown in [116] that around 90% of streaming applications can be modeled as acyclic SDF graphs. Once a cycle exists in an SDF graph, one can always fuse all actors in the cycle into a single stateful actor. A stateful actor is the one whose next execution depends on the current execution. As a consequence, our approach does not unfold stateful actors. Furthermore, the data source and sink actors, which represent the external environment, are not unfolded. The target platform assumed in this work is a homogeneous programmable MPSoC with distributed memory. The interconnection structure between PEs must provide guaranteed communication latency, e.g., Æthereal network-on-chip [53].

as a special case of unfolding.

5.1 Related Work

The approach in [113] is closely related to our work, although the considered problem is relaxed, i.e., without considering timing constraints, compared to our problem. A genetic algorithm based heuristic is proposed to determine the unfolding factor of an actor and allocation of all replicas. The unfolding factor of an actor cannot exceed the number of PEs, which might result in sub-optimal solutions as we show later in Section 5.5. Moreover, we show in the experiments that our approach outperforms significantly the genetic algorithm based heuristic in terms of running time.

In [71], an Integer Linear Programming (ILP) formulation gives exact solutions to minimize makespan on any PE while simultaneously unfolding actors in an SDF graph and allocating them to PEs. In the ILP formulation, an unfolding factor of an actor cannot exceed the number of available PEs. This assumption might lead to sub-optimal system performance as discussed previously. Moreover, it has been shown in [40] that the ILP formulation is even intractable for benchmarks with medium graph size. For instance, it takes around 70 hours to solve the ILP formulation for the FFT benchmark with 26 actors on 4 PEs (see Table 2 in [40]). In practice, real-life applications have been shown to contain up to 2868 actors [116]. Therefore, it is clear that the ILP-based approach suffers from severe scalability issues. In contrast, our proposed algorithm solves the combined problem within a reasonable amount of time as demonstrated later in Section 5.7.

To address the scalability issue of [71], the authors in [40] propose to decompose the actor unfolding and allocation problem into two problems and solve them separately. The separation of the two problems often leads to inferior performance, as both problems are strongly related. In contrast, our proposed algorithm is capable of solving the two problems simultaneously. Moreover, our algorithm takes into account timing constraints, while the work in [40] does not.

In the context of synthesizing an SDF graph using dedicated hardware, the authors in [56] also determine which actors to unfold and by what factor. The addressed problem is easier than ours because there is no need to consider allocation of actors after unfolding in case of hardware synthesis.

In [72], a synchronous programming model is used for the application specification under hard real-time scheduling. The term "synchronous" in this context refers to the fact that a master thread can *fork* a job into several parallel execution segments and they *join* upon completion. These parallel execution segments are, to some extent, similar to unfolded actors in our case. There is also no need to consider allocation of parallel segments at compile-time because migration at run-time is allowed targeting MPSoC platforms with shared memory. In contrast, we solve the problem of allocating actors at compile-time. Recall that we consider MPSoC

platforms with distributed memory. On such platforms, migration of actors at run-time introduces non-negligible overhead.

5.2 Unfolding of SDF Graphs

Before the problem formulation, we first present an unfolding algorithm for SDF graphs. This will help better understand the problems stated in Section 5.3.

The unfolding operation on an SDF graph used in this thesis is conceptually similar to the one used in [40, 54, 56, 71], in which two special constructs *splitter* and *joiner* are employed for the unfolded actors. Given a vector $\vec{f} \in \mathbb{N}^n$ of unfolding factors, where f_i denotes the unfolding factor for actor A_i, the unfolding operation replaces A_i by f_i replicas of itself. Then, instead of inserting a splitter and joiner before and after the f_i replicas of A_i, we transform the initial SDF graph to a functionally equivalent CSDF graph. To ensure the functional equivalence, the production and consumption rates of an SDF actor are modified accordingly to the production and consumption sequences in the resulting CSDF graph. This modification results in a different repetition vector of the obtained CSDF graph to ensure its consistency.

The algorithm for performing the unfolding of actors in SDF graphs is given in Algorithm 5. The algorithm accepts as inputs an SDF graph G and a vector $\vec{f}$ of unfolding factors. The algorithm produces as an output a CSDF graph G', where $A_{i,f}$ denotes the fth replica of A_i with repetition $q_{i,f}$ given by

$$q_{i,f} = \frac{q_i \cdot \text{lcm}(\vec{f})}{f_i}, \tag{5.1}$$

where q_i is the repetition of actor A_i in the initial SDF graph and $\text{lcm}(\vec{f})$ denotes the least common multiple of $f_i \in \vec{f}$. It follows that the repetition vector of G', denoted by $\vec{q'} \in \mathbb{N}^{n'}$ where $n' = \sum_{A_i \in \mathcal{A}} f_i$, is given by $\vec{q'} = [q_{1,1}, \cdots, q_{1,f_1}, \cdots, q_{n,f_n}]^T$ and $n = |\mathcal{A}|$. After obtaining $\vec{q'}$ using Equation 5.1, production/consumption sequences of each CSDF actor are generated accordingly.

Let us consider an SDF graph G_1 is shown in Figure 5.1(a). The actors A_1 and A_5 are the data source and sink actors, respectively. G_1 has five actors and a repetition vector $\vec{q} = [1, 1, 2, 1, 1]^T$. The WCET of each actor is shown below its name, e.g., $C_3 = 12$ for actor A_3. Suppose that a vector of unfolding factors is given as $\vec{f} = [1, 1, 3, 1, 1]$ for G_1 in Figure 5.1(a). Algorithm 5 outputs a CSDF graph G_2 shown in Figure 5.1(b) with three replicas $A_{3,1}$, $A_{3,2}$ and $A_{3,3}$ for actor A_3 in G_1. The

Algorithm 5: Unfolding an SDF graph.

Input: An SDF graph $G = (\mathcal{A}, \mathcal{E})$ and a vector $\vec{f}$ of unfolding factors.
Result: The equivalent CSDF graph $G' = \{\mathcal{A}', \mathcal{E}'\}$

$\mathcal{A}' = \emptyset, \mathcal{E}' = \emptyset$;
foreach $A_i \in \mathcal{A}$ **do**
    Add $f_i \in \vec{f}$ replicas of A_i to $\mathcal{A}'$;
    Set repetition entry $q_{i,ii} = \frac{q_i \cdot \mathrm{lcm}(\vec{f})}{f_i}, \forall ii \in [1, f_i]$;
foreach $E \in \mathcal{E}$ **do**
    Get source actor A_i and sink actor A_j of edge E ;
    Get production rate $prd(E)$ and consumption rate $cns(E)$;
    $lcm\_pc = \mathrm{lcm}(prd(E), cns(E))$;
    if f_j *is dividable by* f_i **then** $OP = f_j/f_i; IP = 1;$
    else if f_i *is dividable by* f_j **then** $IP = f_i/f_j; OP = 1;$
    else $IP = f_i/f_j; OP = 1;$
    for $ii = 1$ **to** f_i **do**
        Add OP output ports to $A_{i,ii}$;
        for $k = 1$ **to** OP **do**
            Initialize a production sequence $\mathcal{P}_{i,ii}$ of length $q_{i,ii}$ to 0;
            $\mathcal{P}_{i,ii}[p] = prd(E), \forall p \in [(k-1)\frac{lcm\_pc}{prd(E)} + 1, k\frac{lcm\_pc}{prd(E)}]$;
            if f_j *is dividable by* f_i **then** $jj = (ii-1)OP + k$;
            else if f_i *is dividable by* f_j **then** $jj = ii/IP$;
            else $jj = k$;
            Initialize a consumption sequence $\mathcal{C}_{j,jj}$ of length $q_{j,jj}$ to 0;
            $\mathcal{C}_{j,jj}[c] = cns(E), \forall c \in [(ii-1)\frac{lcm\_pc}{cns(E)} + 1, ii\frac{lcm\_pc}{cns(E)}]$;
            Create a new channel E' connecting replica $A_{i,ii}$ to replica $A_{j,jj}$;
            Add channel E' to $\mathcal{E}'$;
Compact the production and consumption sequences of each actor in $\mathcal{A}'$;

unfolding results in a repetition vector of G_2 as:

$$\begin{aligned}\vec{q'}_{G_2} &= [q_{1,1}, q_{2,1}, q_{3,1}, q_{3,2}, q_{3,3}, q_{4,1}, q_{5,1}]^T \\ &= [3,3,2,2,2,3,3]^T\end{aligned}$$

For example, SDF actor A_4 executes only once ($q_4 = 1$) in G_1 per graph iteration,

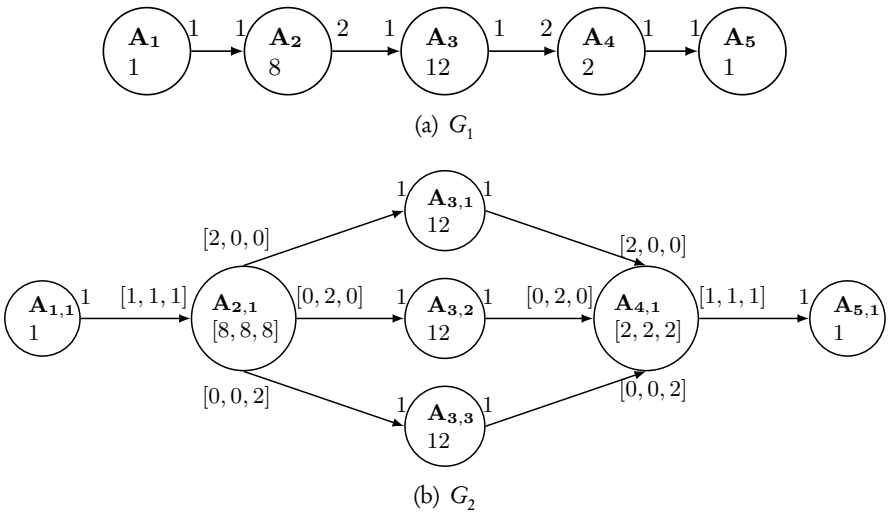

Figure 5.1: (a) An example of an SDF graph and (b) its equivalent CSDF graph by unfolding actor A_3 by factor 3.

while executing three times ($q_{4,1} = 3$) in G_2 per graph iteration. Three consumption sequences of actor $A_{4,1}$ in G_2 behave similar to a joiner, with which $A_{4,1}$ collects data tokens from the three replicas $A_{3,1}$, $A_{3,2}$ and $A_{3,3}$. Analogous to a splitter, actor $A_{2,1}$ with three production sequences distributes tokens to the three replicas.

5.3 Problem Formulation

First of all, recall the notations used for (C)SDF MoCs in Table 2.3 on page 31 and the notations used for HRT scheduling of (C)SDF MoCs in Table 2.4. In addition, we introduce some extra notations in Table 5.1 used in this chapter to facilitate the following discussion. Let T_i be the actual period of actor $A_i \in \mathcal{A}$ of a CSDF graph $G = (\mathcal{A}, \mathcal{E})$. T_i can be obtained as $T_i = c\check{T}_i$, where $\check{T}_i$ is computed in Equation (2.16) on page 34 and c is called *scaling factor* (see Section 5.4). Now, we formally define our problem as follows:

Problem 5.3.1. Given an SDF graph G, where the actors are scheduled as strictly periodic tasks, and m available PEs. Suppose that each actor A_i in G is to be unfolded by an unfolding factor $f_i \in \mathbb{N}^+$. Find, for each actor A_i, the minimum value of f_i and the allocation of each replica $A_{i,d}$, where $1 \leq d \leq f_i$, such that the period of the sink actor T_{snk} (see definition of T_{snk} on page 34) in the unfolded graph is minimized.

If Problem 5.3.1 is considered as primal, its dual problem can be stated as follows:

Problem 5.3.2. Given an SDF graph G, where the actors are scheduled as strictly periodic tasks, and m available PEs. Suppose that each actor A_i in G is to be unfolded by an unfolding factor f_i. Find, for each actor A_i, the minimum value of f_i and the allocation of each replica $A_{i,d}$, where $1 \leq d \leq f_i$, such that the total utilization

$$U_{G'} = \sum_{A_{i,d} \in \mathcal{A}'} \frac{C_{i,d}}{T_{i,d}}$$

of the unfolded graph G' on m PEs is maximized, where $T_{i,d}$ is the actual period of replica $A_{i,d}$.

It can be seen that Problems 5.3.1 and 5.3.2 are not trivial. In general, for a given SDF graph, the number of possible alternative graphs that can be generated using unfolding grows exponentially as the number of actors increases. Furthermore, for each alternative graph, we have to perform allocation of unfolded actors which is by itself an NP-hard problem.

Lemma 5.3.1. *Problems 5.3.1 and 5.3.2 are equivalent.*

Proof. The lemma is proven by showing that a solution to Problem 5.3.1 is also a solution to Problem 5.3.2 (case I) and vice versa (case II).
Case I:

Let G' be the unfolded graph of G. Suppose that $\vec{f}$ is the solution to Problem 5.3.1. This means that T_{snk} is minimized. The period $T_{i,f}$ of a replica $A_{i,f}$ in G' based on Equation (2.16) on page 34 can be written as

$$T_{i,f} = c \cdot \check{T}_{i,f} = \frac{c \cdot \text{lcm}(\vec{q'})}{q_{i,f}} \left\lceil \frac{\hat{W}_{G'}}{\text{lcm}(\vec{q'})} \right\rceil, \tag{5.2}$$

Notation	Meaning
c	scaling factor for periods of all actors in a (C)SDF graph and $c \in \mathbb{Z}^+$
f_i	unfolding factor for actor A_i
Ω	ratio
ρ	quality factor $\rho \in (0, 1]$
θ_i	code size of a (C)SDF actor A_i
Θ	total code size of a (C)SDF graph, $\Theta = \sum_{A_i \in \mathcal{A}} \theta_i$

Table 5.1: Additional notations used in Chapter 5 besides the ones introduced in Chapter 2.

where c is the scaling factor. Thus, for the sink actor, it is

$$T_{\text{snk}} \cdot q_{\text{snk}} = c \cdot \text{lcm}(\vec{q'}) \left\lceil \frac{\hat{W}_{G'}}{\text{lcm}(\vec{q'})} \right\rceil, \tag{5.3}$$

where $c \cdot \text{lcm}(\vec{q'}) \lceil \hat{W}_{G'} / \text{lcm}(\vec{q'}) \rceil$ is constant. Therefore, from Equations 5.2 and 5.3, it holds

$$T_{\text{snk}} \cdot q_{\text{snk}} = T_{i,f} \cdot q_{i,f}, \forall A_{i,f}. \tag{5.4}$$

Subsequently, Equation 5.4 can be re-written as

$$T_{i,f} = \frac{T_{\text{snk}} \cdot q_{\text{snk}}}{q_{i,f}}. \tag{5.5}$$

Due to the devised unfolding algorithm (see Algorithm 5), we have $q_{i,1} = q_{i,d}$ (see Equation (5.1)), where $1 \leq d \leq f$. Therefore, $\beta_i = q_{\text{snk}}/q_{i,f}$ is constant. It then follows

$$T_{i,f} = \beta_i T_{\text{snk}} \tag{5.6}$$

It follows from Equation 5.6 that when T_{snk} is minimized, then $T_{i,f}$ is minimized. Recall that the maximum total utilization $\hat{U}_{G'}$ is given by

$$\hat{U}_{G'} = \sum_{A_{i,f} \in \mathcal{A}} \frac{C_{i,f}}{T_{i,f}} \tag{5.7}$$

Since $T_{i,f}$ is minimized for all the actors, it follows that $\hat{U}_{G'}$ is maximized. Therefore, $\vec{f}$ is also the solution to Problem 5.3.2.

Case II:

Suppose that $\vec{f}$ is the solution to Problem 5.3.2. This means that $\hat{U}_{G'}$ is maximized. Using Equation 5.4, we can replace each $T_{i,f}$ in Equation 5.7 by $\frac{T_{\text{snk}} \cdot q_{\text{snk}}}{q_{i,f}}$, which results in:

$$\hat{U}_{G'} = \frac{C_{1,1} \cdot q_{1,1}}{T_{\text{snk}} \cdot q_{\text{snk}}} + \cdots + \frac{C_{2,1} \cdot q_{2,1}}{T_{\text{snk}} \cdot q_{\text{snk}}} + \cdots + \frac{C_{\text{snk}}}{T_{\text{snk}}} \tag{5.8}$$

The WCET $C_{i,f}$ and repetition $q_{i,f}$ of each replica is constant. Therefore, Equation 5.8 can be re-written as:

$$\hat{U}_{G'} = \frac{\alpha_{1,1}}{T_{\text{snk}}} + \cdots + \frac{\alpha_{2,1}}{T_{\text{snk}}} + \cdots + \frac{\alpha_{\text{snk}}}{T_{\text{snk}}} \tag{5.9}$$

where $\alpha_{i,f} = C_{i,f} \cdot q_{i,f}/q_{\text{snk}}$. Since $\hat{U}_{G'}$ is maximized, it follows that T_{snk} is minimized. Therefore, $\vec{f}$ is also a solution to Problem 5.3.1. ■

5.4 Period Scaling under Hard Real-time Scheduling

As given in Equation (2.16) on page 34, a (C)SDF graph under SPS can achieve the minimum period (inverse of maximum throughput) $\check{T}_i$ for each actor A_i. This maximum performance also requires the largest number of PEs. If the maximum performance is not necessary, the actually desired period T_i of A_i can be obtained by scaling up $\check{T}_i$ by a scaling factor $c \in \mathbb{Z}^+$ as $T_i = c\check{T}_i$. Using a scaling factor c, we can have a trade-off between processing resources and guaranteed performance as shown in the following proposition:

Proposition 5.4.1. *Let G be a CSDF graph that is schedulable using a scheduling algorithm SA and an allocation algorithm AA on $\check{m}$ PEs, when the minimum period of each actor A_i is equal to $\check{T}_i$. G is schedulable using the same SA and AA on $\lceil \frac{\check{m}}{c} \rceil$ PEs, when the period of each actor A_i is scaled by c.*

Proof. Let U_{SA} be the utilization bound of a scheduling algorithm SA. If G is schedulable on $\check{m}$ PEs using SA and any AA, then this means that the total utilization of the actors on each PE j, where $1 \leq j \leq \check{m}$, is $U_{\mathrm{PE}_j} \in (0, U_{SA}]$. If we scale the periods of the actors in G by c, then this means that $U_{\mathrm{PE}_j} \in (0, \frac{U_{SA}}{c}]$. Therefore, it is possible to combine the actors in every c PEs into 1 PE. Hence, the number of PEs needed after scaling the periods is $\lceil \frac{\check{m}}{c} \rceil$. ■

Considering G_2 in Figure 5.1(b), we have computed in Equation (2.24) on page 37 that it can be scheduled on 5 PEs while achieving $\vec{\check{T}}_{G_2}$. Therefore, it can be scheduled on $\lceil \frac{5}{2} \rceil = 3$ PEs achieving a period $T_{5,1} = 2 \times \check{T}_{5,1} = 16$, i.e., throughput $\frac{1}{16}$ by scaling all minimum periods by $c = 2$.

Now, suppose that AA is an approximate allocation algorithm with an approximation ratio R_{AA}. Then, we can have the following proposition:

Proposition 5.4.2. *Let G be a CSDF graph that is schedulable using a scheduling algorithm SA and any exact allocation algorithm on $\check{m}$ PEs, when the period of each actor A_i is equal to $c\check{T}_i$. G is schedulable using SA and any approximate allocation algorithm AA, with approximation ratio R_{AA}, on $\check{m}$ PEs, when the period of each actor A_i is equal to $cR_{AA}\check{T}_i$.*

5.5 Bounding Solution Space

In order to solve Problems 5.3.1 and 5.3.2 defined in Section 5.3, it is first necessary to show that the solution space of the problems is bounded, i.e., the values of the

unfolding factors f_i must be bounded by finite integers. Bounding the solution space ensures that the algorithm devised in Section 5.6 terminates. Now, we define the *upper bound* on unfolding factors as follows:

Definition 5.5.1 (Upper bounds of Unfolding Factors)**.** Let G be an SDF graph, where the actors in G are under SPS shown in Section 2.3, and assume that the number of PEs is unlimited. Suppose that every actor A_i in G is to be unfolded by a factor f_i resulting in a CSDF graph G', for which $\check{T}_{i,f}$ is the minimum period of each replica $A_{i,f}$ and $C_{i,f} = C_i$ is its WCET. The *upper bound* on f_i, denoted by $\hat{f}_i$, is the minimum value which results in utilization

$$\frac{C_{i,f}}{\check{T}_{i,f}} = 1.0$$

for each replica $A_{i,f}$ in G'.

In other words, unfolding an SDF graph G by a vector of unfolding factors $\vec{\hat{f}} = [\hat{f}_1, \cdots, \hat{f}_n]$ results in a graph G' with utilization $U_{G'} = n'$, where n' is the number of actors in the unfolded graph. Hence, unfolding any actor A_i by an unfolding factor $f_i^* > \hat{f}_i$ cannot result in any increase in the total utilization of the unfolded graph. Moreover, the unfolded graph achieves the maximum achievable throughput since the sink actor fully utilizes the PE on which it executes. Therefore, $\vec{\hat{f}}$ bounds the solution space that has an impact on the total utilization of the unfolded graph.

Determining the upper bound $\vec{\hat{f}}$, however, is not trivial. One common assumption, e.g., in [54] and [71], is to set $\vec{\hat{f}} = [m, m, \cdots, m]$, where m is the number of PEs. In this section, we show, using an example, that this assumption may limit the solution space. As a consequence, the limited solution space might not contain the optimal solution to Problems 5.3.1 and 5.3.2.

Let us consider G_1 in Figure 5.1(a) and suppose that 2 PEs are available. The optimal alternative graph of G_1 is G_3, shown in Figure 5.2, when the vector of unfolding factors is $\vec{f} = [1, 2, 4, 1, 1]$. First, the repetition vector of G_3 can be computed according to Equation (5.1) as

$$\begin{aligned}\vec{q}_{G_3} &= [q_{1,1}, q_{2,1}, q_{2,2}, q_{3,1}, q_{3,2}, q_{3,3}, q_{3,4}, q_{4,1}, q_{5,1}] \\ &= [4, 2, 2, 2, 2, 2, 2, 4, 4].\end{aligned}$$

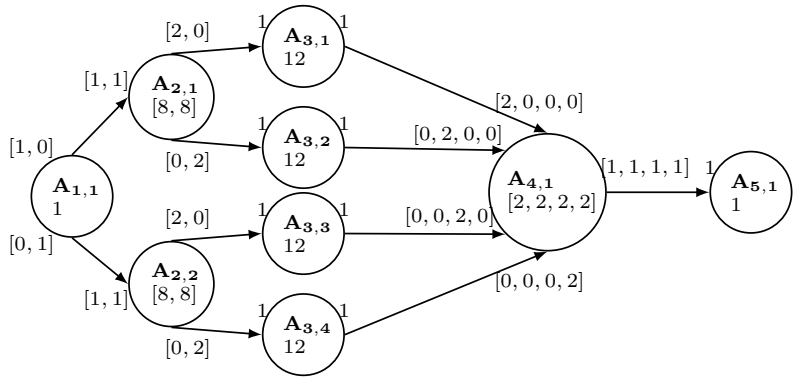

Figure 5.2: G_3: Optimal alternative graph of G_1 in Figure 5.1(a) with unfolding factors $f_2 = 2, f_3 = 4$ when scheduled on 2 PEs.

It follows that $\hat{W}_{G_3} = q_{3,1} \times C_{3,1} = 2 \times 12$ and $\text{lcm}(\vec{q}_{G_3}) = 4$. Solving Equation (2.16) on page 34 yields the minimum period of the sink actor $A_{5,1}$ as

$$\check{T}_{snk} = \frac{4}{4} \cdot \lceil \frac{24}{4} \rceil = 6.$$

To achieve $\check{T}_{\text{snk}} = 6$, it requires 6 PEs. Then, we can scale all periods of the actors in G_3 by $c = 3$, which yields a period $T_{\text{snk}} = 3\check{T}_{\text{snk}} = 18$. According to Proposition 5.4.1, the graph G_3 is schedulable on $\lceil \frac{6}{3} \rceil = 2$ PEs. After scaling the periods of all actors, the total utilization U_{G_3} of G_3 on 2 PEs is $U_{G_3} = 2.0$. Since Lemma 5.3.1 states that the maximum utilization corresponds to the minimum period that a CSDF graph can achieve, no shorter period can be achieved for G_3. Thus, G_3 is the optimal alternative graph of G_1 for 2 PEs with an unfolding factor $f_3 = 4$, which is greater than the number of PEs available. Therefore, this example shows that the optimal solution is beyond $\hat{\vec{f}} = [2,2,2,2,2]$, which defines the solution space if we set $\hat{\vec{f}} = [m,m,m,m,m]$. Hence, we conclude that the upper bound on an unfolding factor is not necessarily equal to the number of PEs.

Now, we derive the upper bound on the unfolding factor for each actor in the initial SDF graph by stating the following theorem:

Theorem 5.5.1. *Given an SDF graph G under SPS, suppose that each actor A_i is to be unfolded by a factor f_i. The upper bound on f_i according to Definition 5.5.1 can be computed as follows:*

$$\hat{f}_i = \frac{\text{lcm}\{x_1, x_2, \cdots, x_n\}}{x_i}, \tag{5.10}$$

where

$$x_i = \frac{\text{lcm}\{W_1, W_2, \cdots, W_n\}}{W_i}. \tag{5.11}$$

W_i is the workload of actor A_i given in Definition 2.3.2 on page 34.

Proof. Suppose that G' is the CSDF graph obtained by unfolding each actor A_i in the initial SDF graph G by $\hat{f}_i$. From Definition 5.5.1, it follows that every replica $A_{i,f}$ in G' has $\check{T}_{i,f} = C_{i,f} = C_i$. Therefore, we can re-write Equation 2.16 on page 34 as:

$$C_i = \frac{\text{lcm}(\vec{q'})}{q_{i,f}} \left\lceil \frac{\hat{W}_{G'}}{\text{lcm}(\vec{q'})} \right\rceil \tag{5.12}$$

where $q_{i,f}$ is the repetition of $A_{i,f}$ in G'. Equation 5.12 can be re-written as:

$$q_{i,f} C_i = \text{lcm}(\vec{q'}) \left\lceil \frac{\hat{W}_{G'}}{\text{lcm}(\vec{q'})} \right\rceil \tag{5.13}$$

Since $\text{lcm}(\vec{q'}) \lceil \hat{W}_{G'} / \text{lcm}(\vec{q'}) \rceil$ is constant, then we re-write Equation 5.13 as:

$$q_{1,1} C_1 = q_{1,2} C_1 = \ldots = q_{1,f_1} C_1 = \ldots = q_{n,f_n} C_n \tag{5.14}$$

Now, we can write $q_{i,f} = x_i \cdot q_i$, where q_i is the repetition of A_i in the initial SDF graph and x_i is an integer factor. That is:

$$x_1 q_1 C_1 = x_2 q_2 C_2 = \cdots = x_n q_n C_n \tag{5.15}$$

Equation 5.15 can be re-written as:

$$x_1 W_1 = x_2 W_2 = \cdots = x_n W_n \tag{5.16}$$

where W_i is the workload of actor A_i according to Definition 2.3.2 on page 34. The minimum solution to Equation 5.16 is:

$$x_i = \frac{\text{lcm}\{W_1, W_2, \cdots, W_n\}}{W_i} \tag{5.17}$$

Since $q_{i,f} = x_i q_i$ and the graph is unfolded by $\vec{\hat{f}}$, we can substitute this in Equation 5.1 to get:

$$x_i q_i = \frac{q_i \, \text{lcm}(\vec{\hat{f}})}{\hat{f}_i} \tag{5.18}$$

which can be re-written as:

$$x_i \hat{f}_i = \text{lcm}(\vec{\hat{f}}) \tag{5.19}$$

Since $\text{lcm}(\vec{\hat{f}})$ is constant, Equation 5.19 can be re-written as:

$$x_1 \hat{f}_1 = x_2 \hat{f}_2 = \cdots = x_n \hat{f}_n \tag{5.20}$$

The minimum solution to Equation 5.20 is:

$$\hat{f}_i = \frac{\text{lcm}\{x_1, x_2, \cdots, x_n\}}{x_i} \tag{5.21}$$

■

Now, we give an example on how to compute $\vec{\hat{f}}$. For G_1 in Figure 5.1(a), we first compute $\text{lcm}\{W_1, W_2, W_3, W_4, W_5\} = 24$. Then, $\vec{x}$ containing the values of x_i is given by

$$\begin{aligned}\vec{x} &= [x_1, x_2, x_3, x_4, x_5] \\ &= [\frac{24}{1}, \frac{24}{8}, \frac{24}{24}, \frac{24}{2}, \frac{24}{1}] \\ &= [24, 3, 1, 12, 24].\end{aligned}$$

Finally we obtain $\text{lcm}(\vec{x}) = 24$, and

$$\begin{aligned}\vec{\hat{f}} &= [f_1, f_2, f_3, f_4, f_5] \\ &= [\frac{24}{24}, \frac{24}{3}, \frac{24}{12}, \frac{24}{12}, \frac{24}{24}] \\ &= [1, 8, 24, 2, 1].\end{aligned}$$

5.6 The Algorithm

Based on the upper bounds on unfolding factors, we devise, in this section, an efficient algorithm to solve Problems 5.3.1 and 5.3.2 under a given number of PEs.

The algorithm accepts as an input the following:

1. the initial SDF graph G;
2. the number of available PEs m;

3. the vector containing the upper bounds on the unfolding factors $\vec{\hat{f}}$ computed using Equation 5.10;

4. a pre-specified quality factor $\rho \in (0,1]$, which is used to terminate the algorithm. $\rho = 1$ indicates the highest quality that m PEs must be fully utilized when allocating the resulting graph to the m PEs.

The outputs of the algorithm are:

1. a vector of unfolding factors that is the solution to Problems 5.3.1 and 5.3.2;

2. the allocation of the unfolded SDF graph on m PEs;

3. the minimum achievable period of the sink actor in the unfolded SDF graph on m PEs which is the objective of Problem 5.3.1;

4. the maximum utilization of the unfolded SDF graph on m PEs which is the objective of Problem 5.3.2.

The algorithm builds, incrementally during its execution, a list of nodes in which each node represents a possible vector of unfolding factors $\vec{f}$. Initially, the list contains only a single node which corresponds to the given initial SDF graph with a vector of unfolding factors $\vec{f} = \vec{1}$. Then, we compute the minimum period of the sink actor T_{snk} in the initial SDF graph G, when G is allocated on m PEs, and its total utilization U_G. Both values initialize a tuple $(T_{\text{best}}, U_{\text{best}})$ which holds the period and total utilization of the current best solution. During the execution of the algorithm, new nodes are created and added to the list, where a node represents an alternative CSDF graph G' of the initial graph G with a vector $\vec{f}$ of unfolding factors. Each entry $f_i \in \vec{f}$ ranges from 1 up to $\hat{f}_i$ derived in Equation (5.10).

A newly created node inherits from its previous node a copy of the unfolding factors vector $\vec{f}_{\text{prev}}$ used by the previous node to generate the unfolded graph G'_{prev}. After that, we search in G'_{prev} for the bottleneck actor, denoted by $A_{b,f}$, which is the one with the maximum workload $\hat{W}_G$ as defined in Definition 2.3.2 on page 34. If multiple actors have the same maximum workload, then the one with the smallest code size is selected. Next, we increment by one the entry f_b in the inherited unfolding factors vector $\vec{f}_{\text{prev}}$, thereby, obtaining $\vec{f}_{\text{curr}}$. Then, we unfold the initial graph G by the factors in $\vec{f}_{\text{curr}}$ which results in a CSDF graph G'_{curr}. The next step is to evaluate the unfolded graph G'_{curr} when it is allocated on m PEs. The procedure for evaluating G'_{curr} is explained in details later. The result of the evaluation procedure is the minimum period of the sink actor T_{snk} in G'_{curr}, when G'_{curr} is allocated on m

PEs, and the total utilization of the graph U_{curr}. If the obtained U_{curr} is higher than U_{best} corresponding to the current best solution (i.e., T_{snk} smaller than T_{best}), then T_{best} and U_{best} are updated with T_{snk} and U_{curr}, respectively. Otherwise, T_{best} and U_{best} remain unchanged.

The creation of new nodes is terminated when one of the following conditions is met:

1. The total utilization $U_{G'}$ of the CSDF graph at the current node satisfies $U_{G'} \geq \rho m$, where $\rho \in (0, 1]$ is the quality factor given as an input to the algorithm.

2. The unfolding factor f_i of an actor A_i exceeds either its upper bound $\hat{f}_i$ if A_i is stateless, or 1 if A_i is stateful or a data source/sink actor. Recall that stateful actors together with the data source and sink actors cannot be unfolded.

After the creation of new nodes is terminated, we select the first node in the list that has a minimum sink period and a total graph utilization equal to T_{best} and U_{best}, respectively. The selected node contains the solution to Problems 5.3.1 and 5.3.2.

Evaluating Unfolded Graphs

As explained previously, at each node, the initial SDF graph G is unfolded to produce a CSDF graph $G' = (\mathcal{A}', \mathcal{E}')$. Then, we compute two values for G', i.e.,

1. the minimum sink actor period T_{snk} when G' is allocated on m PEs;

2. its total utilization $U_{G'}$.

In this section, we explain in detail how these two values are computed. Recall from Section 5.4 that T_{snk} can be scaled by a scaling factor c given by $T_{\text{snk}} = c\check{T}_{\text{snk}}$, and $U_{G'}$ can be computed as follows:

$$U_{G'} = \sum_{A_{i,f} \in \mathcal{A}'} \frac{C_{i,f}}{c \cdot \check{T}_{i,f}}. \tag{5.22}$$

Recall also that the objective of Problem 5.3.2 is to maximize the utilization. Therefore, we need to find a value of scaling factor c, such that all actors $A_{i,f} \in G'$ are schedulable on m PEs and $U_{G'}$ is maximized. To do so, we first bound the search range for c by deriving its lower and upper bounds. Using any allocation algorithm, we have from Proposition 5.4.1 a lower bound on c, denoted by $\check{c}$, as follows:

$$\check{c} = \left\lceil \frac{1}{m} \sum_{A_{i,f} \in \mathcal{A}'} \frac{C_{i,f}}{\check{T}_{i,f}} \right\rceil. \tag{5.23}$$

Algorithm 6: The procedure for evaluating an unfolded graph.

Input: A CSDF graph G', number of available PEs m, and the period and total utilization corresponding to the current best solution T_{best} and U_{best}.

Result: *alloc* which is an m-partition describing the allocation of the actors in G' onto m PEs

```
1  Compute č using Equation 5.23 and ĉ using Equation 5.24 ;
2  for s = č to ĉ do
3      Compute the period T_{i,f} of each actor A_{i,f} as T_{i,f} = c·Ť_{i,f} ;
4      if T_snk ≥ T_best then
5          return ∅ ;
6      Compute the utilization U_{G'} using Equation 5.22;
7      Find an m'-partition of the actors in G', denoted by alloc, using the FFD
       algorithm and assuming the EDF scheduling algorithm;
8      if m' ≤ m then
9          U_best = U_{G'}, T_best = T_snk;
10         return alloc ;
```

That is, for any AA, the scaling factor c cannot be less than $\check{c}$. From Proposition 5.4.2, we compute, using the approximation ratio of the First-Fit Decreasing (FFD) allocation algorithm given in Equation (2.23) on page 37, the upper bound on the scaling factor c, denoted by $\hat{c}$, as follows:

$$\hat{c} = \left\lceil \frac{11}{9m} \sum_{A_{i,f} \in \mathcal{A}'} \frac{C_{i,f}}{\check{T}_{i,f}} \right\rceil + 1. \tag{5.24}$$

Once the lower and upper bounds of c are found using Equation (5.23) and Equation (5.24), respectively, we perform a linear search to seek the smallest c, such that CSDF graph G' is schedulable on m PEs. Specifically, we check if an m-partition of all actors in G' exists, assuming the EDF scheduling algorithm and the FFD allocation algorithm explained in Section 2.3. The complete procedure for evaluating the unfolded graphs is depicted in Algorithm 6. If the period resulting from a given scaling factor c is greater than T_{best}, then Algorithm 6 terminates immediately to speed-up the search.

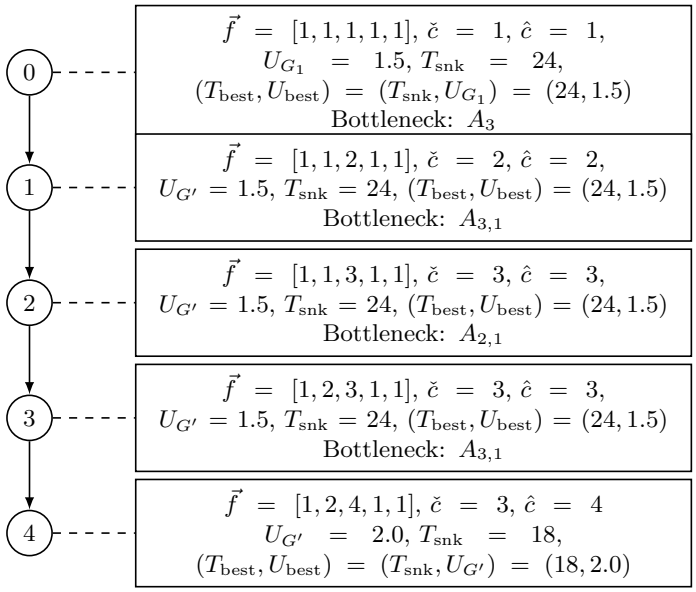

Figure 5.3: The list produced by the algorithm for G_1 in Figure 5.1(a) on 2 PEs with $\rho = 0.95$.

Example

Now, we illustrate our algorithm using graph G_1 in Figure 5.1(a) and schedule the resulting graph G' on 2 PEs (i.e., $m = 2$) with the EDF algorithm. Suppose that $\rho = 0.95$, i.e., the algorithm terminates when $U_{G'} \geq 0.95 \times 2 = 1.9$. The whole list produced by the algorithm is illustrated in Figure 5.3. The numbers inside the nodes correspond to the sequence in which the nodes are created. The algorithm starts with the initial G_1 in node 0 and computes the scaling factors $\check{c}$ and $\hat{c}$ which result in $U_{G_1} = 1.5$ and period $T_{\text{snk}} = 24$. At this point, U_{best} is initialized to 1.5 and T_{best} to 24. Node 1 inherits from node 0 a vector of unfolding factors equal to $[1,1,1,1,1]$. After that, we search in $G'_{\text{prev}} = G_1$ for the bottleneck actor which is A_3. Next, we increment f_3 in the inherited vector of unfolding factors at node 1 resulting in $\vec{f} = [1,1,2,1,1]$. Then, G' is generated and Algorithm 6 is invoked. Since U_{best} cannot be improved, the algorithm continues by creating node 2. At node 2, a new bottleneck actor $A_{2,1}$ is introduced. Therefore, at node 3, the unfolding factor f_2 is incremented by 1. Then, the algorithm continues to node 4, at which one termination criterion is met, namely $U_{G'} \geq 1.9$. As a result, $\vec{f} = [1,2,4,1,1]$ is the solution with $T_{\text{best}} = 18$ and $U_{\text{best}} = 2.0$.

5.7 Experimental Evaluation

In this section, we present the results of evaluating our algorithm presented in Section 5.6 using a set of real-life streaming applications. We evaluate the algorithm by performing two experiments. In the first experiment, we run our algorithm on the applications and report the following:

1. the performance gain resulting from mapping the SDF graph unfolded using the unfolding factors obtained from our algorithm, compared to mapping the initial SDF graph without unfolding;

2. the total time needed to execute our algorithm.

In the second experiment, we compare our proposed algorithm with one of the state-of-the-art search meta-heuristics because Problems 5.3.1 and 5.3.2 in general can be readily formulated and solved by these meta-heuristics, such as genetic algorithms, simulated annealing, etc. However, meta-heuristics normally require parameter tuning to achieve a good solution. In this work, we select a particular meta-heuristic, namely Genetic Algorithms (GA) for two reasons:

1. they have been applied by several researchers to solve similar problems (e.g., [113]);

2. several researchers have reported the optimal parameter settings for GA in the context of our problem (e.g., [118]).

In particular, we compare our proposed algorithm with the one based on the NSGA-II genetic algorithm [36]. Specifically, we compare two metrics:

1. the total execution time needed by each algorithm to find a solution;

2. the total code size of the returned solution.

We conducted all experiments on 11 real-life streaming applications from the StreamIt benchmarks suite [54]. The exact characteristics of the benchmarks are outlined in Table 5.2. Overall, the number of actors in the benchmarks varies from 8 to 120. The WCET of each actor was profiled on the RAW achitecture. The benchmarks include two applications with stateful actors, namely MPEG2 and Vocoder. Both our algorithm and the meta-heuristic were developed in the Phrt tool as part of Daedalus^{RT} shown in Figure 1.7 on page 13. For the NSGA-II GA, we used the implementation from the DEAP [44] framework. All experiments were performed on an Intel Core 2 Duo T9600 CPU running at 2.80 GHz with Linux Kubuntu 10.4.

Table 5.2: Benchmark characteristics.

Benchmark	Num. of Actors	Num. of Edges	Has Stateful Actors?
DCT	8	7	No
FFT	17	16	No
Filterbank	85	99	No
TDE	29	28	No
DES	53	60	No
Serpent	120	128	No
Bitonic	40	46	No
MPEG2	23	26	Yes
Vocoder	114	147	Yes
FMRadio	43	53	No
Channel	55	70	No

Evaluating the Proposed Algorithm

First, we present the performance gain resulting from mapping the unfolded SDF graph, compared to mapping the initial SDF graph without unfolding. We do so by running the algorithm on the benchmarks and mapping each application on a number of PEs that varies from 2 up to 128 PEs. We evaluate the trade-off between the performance gain and total execution time by setting different quality factors $\rho \in \{0.8, 0.85, 0.9, 0.95\}$. To measure the performance gain, we compute, for each benchmark, the ratio between the sink actor period resulting from mapping the unfolded SDF graph, and the period resulting from mapping the initial SDF. This ratio is denoted by Ω and is given by

$$\Omega = \frac{T_{\text{snk}} \text{ of } G'}{T_{\text{snk}} \text{ of } G},$$

where G' is the unfolded graph, and G is the initial SDF graph. A lower value of Ω indicates a shorter sink actor period in the unfolded graph, and therefore, a higher throughput. In Figure 5.4, each vertical line shows the variations in Ω for all the benchmarks. The marker at the middle of each vertical line represents the Geometric Mean (GM) of Ω, while the upper and lower ends of the line represent the maximum and minimum values of Ω, respectively. It can be seen that mapping the unfolded SDF graphs of the benchmarks achieves significant performance improvement compared to mapping the initial SDF graphs of the benchmarks. As the number of PEs increases, the unfolded SDF graphs utilize the PEs much better than the initial SDF

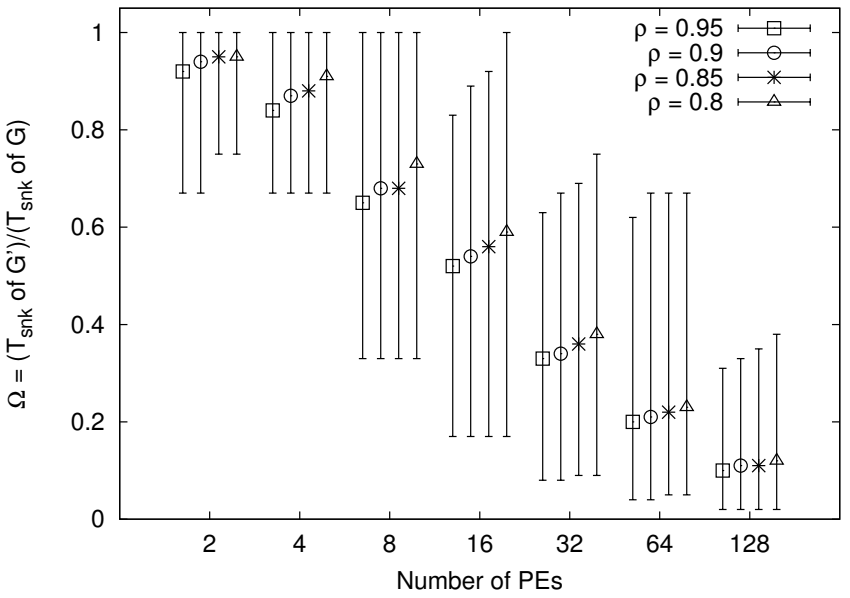

Figure 5.4: Period ratio (lower is better).

graphs. For example, on 64 and 128 PEs, mapping the unfolded SDF graphs with quality factor $\rho = 0.95$ achieves a GM of Ω equal to 0.2 and 0.1, respectively. The DCT benchmark benefits significantly from the algorithm and achieves a GM of Ω equal to 0.021 and 0.042 on 128 and 64 PEs, respectively. Even when a small number of PEs is available, the unfolded SDF graphs still achieve, with quality factor $\rho = 0.95$, a GM of Ω equal to 0.92 and 0.85 on 2 and 4 PEs, respectively.

During the experiment, we also find that the unfolding factor of an actor, obtained using our algorithm, is not necessarily equal to the number of PEs. For example, the obtained unfolded SDF graph of the Vocoder benchmark, when mapped onto 8 PEs, requires the RectangularToPolar actor in the initial SDF graph to be unfolded by a factor of 20. This confirms our statement in Section 5.5. With our provable upper bound, our algorithm results in 4% period reduction for this benchmark compared to other approaches, in which the RectangularToPolar actor is only unfolded with factor 8.

We also evaluate the total execution time of our algorithm, denoted by t_{ours}, when it is invoked on the benchmarks. Figure 5.5 shows the total execution time of our algorithm in seconds for all the benchmarks. For all benchmarks, our algorithm takes a GM of 6.07 seconds for 128 PEs with utilization ratio $\rho = 0.95$. The Serpent benchmark (the largest graph size with 120 actors) takes the longest running time

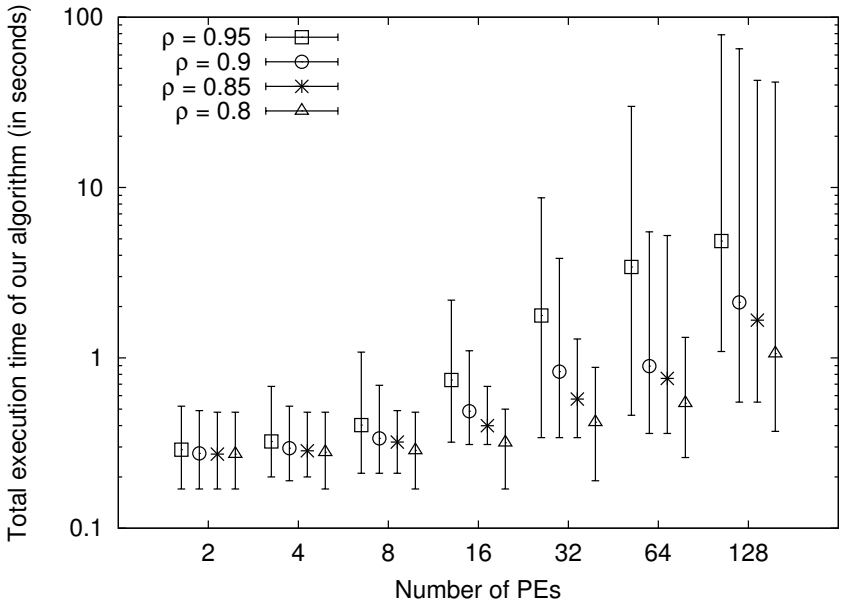

Figure 5.5: Running time of our algorithm.

(78.90 seconds), while the DCT benchmarks takes the shortest running time (1.09 seconds). As the quality factor ρ is decreased from 0.95 to 0.9, the GM of the running time drops to 2.49 seconds for 128 PEs. These results show clearly that our algorithm results, within a reasonable amount of time, in a large performance gain.

Comparison with Genetic Algorithm

To compare our algorithm with the GA-based heuristic, we perform the following steps. First, we run the GA to map each benchmark onto 64 PEs. It outputs an achievable period T and total utilization U_{GA}. Then, we run our algorithm to map the same benchmark onto 64 PEs with a termination criterion $U_{G'} \geq U_{\mathrm{GA}}$. This criterion ensures a fair comparison since our algorithm runs till it finds the same or better solution in terms of the sink actor period and total utilization compared to the best solution found by the GA-based heuristic. Then, we compare two metrics:

1. the total execution time of each algorithm;

2. the total code size Θ resulting from the unfolding factors returned by each

$A_{1,1}$	...	$A_{1,\hat{f}_1}$	...	$A_{n,1}$	...	$A_{n,\hat{f}_n}$
j	...	0	...	1	...	2

Figure 5.6: An example of an individual. The first replica of A_1 is allocated on the jth PE and the $\hat{f}_1$th replica of A_1 does not exist.

algorithm. The total code size is computed as

$$\Theta = \sum_{A_{i,f} \in \mathcal{A}'} \theta_{i,f}$$

where $\theta_{i,f}$ is the code size for actor $A_{i,f}$.

For the GA-based heuristic, each individual (also known as a chromosome) encodes a particular unfolding vector $\vec{f}$ of the initial SDF graph and the allocation of the replicas on m PEs. The structure of an individual is visualized in Figure 5.6. Basically, in an individual, each SDF actor A_i in the initial graph has $\hat{f}_i$ cells as derived in Equation 5.10, indicating that A_i may have up to $\hat{f}_i$ replicas. Each cell may have a value varying from 0 up to m. A value of 0 denotes that the replica does not exist, while a value of 1 up to m denotes the PE on which the replica is allocated. Then, we formulate Problem 5.3.1 as a multi-objective optimization problem with two objectives. The first objective is to minimize the sink actor period, and the second one is to minimize the total code size of the unfolded graph. During the search, we use the evaluation function shown in Algorithm 7. The GA outputs a set of Pareto points, for which we select the one with the shortest achievable period. In order to control the GA, we use the parameters reported in [118], because the target application domain and used platforms are similar to ours. The values of these parameters are given in Table 5.3.

Figure 5.7 shows two ratios. The first ratio (shown in white bars) is the total execution time ratio given by

$$\Omega_t = \frac{t_{\text{GA}}}{t_{\text{ours}}},$$

where t_{GA} is the total time needed by the GA, and t_{ours} is the total time needed by our algorithm. The second ratio in Figure 5.7 (shown in black bars) is the total code size ratio given by

$$\Omega_\Theta = \frac{\Theta_{\text{GA}}}{\Theta_{\text{ours}}},$$

Algorithm 7: Evaluation function in the GA-based meta-heuristic

Input: An individual to be evaluated
Result: An achievable period and total code size.

Check if the given individual is valid ;
if *the individual is invalid* **then return** $(-1, -1)$;
Build the vector of unfolding factors $\vec{f}$ from the individual ;
Generate the CSDF graph G' by unfolding G with $\vec{f}$ using Algorithm 5;
Compute the minimum achievable period $\check{T}_{i,f}$ of each actor $A_{i,f}$ using Equation (2.16) ;
Compute $\check{c}$ according to Equation 5.23 ;
$c = \check{c}$;
while *true* **do**
    Compute the period $T_{i,f}$ of each actor $A_{i,f}$ as $T_{i,f} = c\check{T}_{i,f}$;
    if G' *is schedulable on m PEs* **then**
        Compute total code size $\Theta = \sum_{A_{i,f} \in A'} \theta_{i,f}$;
        Get the period T_{snk} of the sink actor in G' ;
        return $((T_{snk}, \Theta)$;
    else
        $c = c + 1$;

Table 5.3: Parameters for the genetic algorithm.

Parameter	Recommended value in [118]
Population size	80
Number of generations	300
Crossover rate	0.9
Mutation rate	0.05
Mating rate	0.1

where Θ_{GA} is the total code size of the solution obtained using the GA, and Θ_{ours} is the total code size of the solution obtained using our algorithm. Our algorithm is on average 104 times faster than the GA-based heuristic. For example, to unfold and map the FMRadio benchmark onto 64 PEs, our algorithm takes only 3 seconds, while the GA-based heuristic takes 2439 seconds. This means that our algorithm,

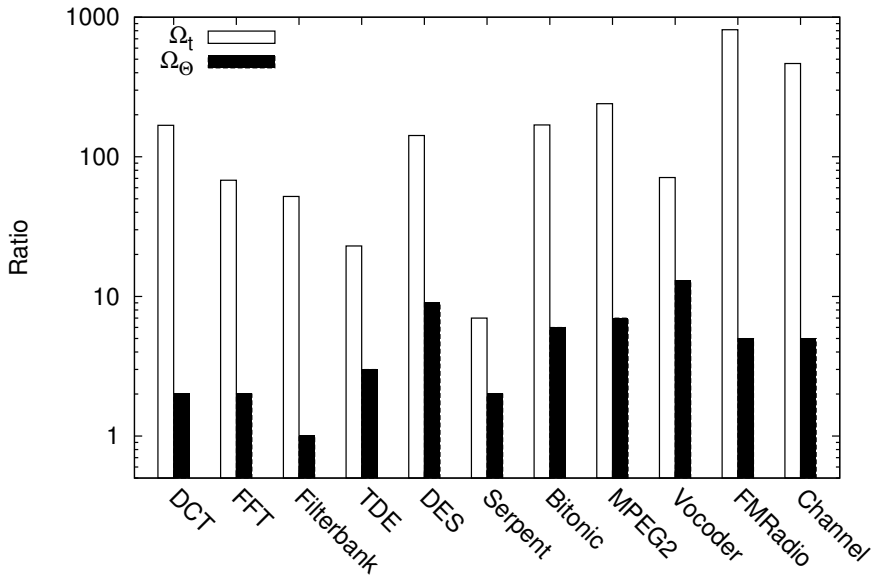

Figure 5.7: The ratios of total execution time Ω_t and total code size Ω_Θ for GA and our algorithm.

for the FMRadio benchmark, is 813 times faster. We also see from Figure 5.7 that our algorithm results in less total code size compared to the GA-based heuristic. These results show clearly that our algorithm outperforms the GA-based heuristic in terms of

1. the time needed to obtain the solution;
2. the total code size of the obtained solution.