Heuristics for scenario creation to enable general loop transformations

Citation for published version (APA):

DOI:
10.1109/ISSOC.2007.4427430

Document status and date:
Published: 01/01/2007

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain.
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Heuristics for Scenario Creation to Enable General Loop Transformations

M. Palkovic¹, H. Corporaal², F. Cattrael⁴
¹IMEC Lab., Kapeldreef 75, 3001 Leuven, Belgium.
⁴Also Professor at Katholieke Universiteit Leuven, Belgium.

Abstract—Embedded system applications can have quite complex control flow structures (CFGs). Often their control flow prohibits design time optimizations, like advanced global loop transformations. To solve this problem, and enable far more global optimizations, we could consider paths of the CFG in isolation. However coding all paths separately would cause a tremendous code copying. In practice we have to trade-off the extra optimization opportunities vs. the code size.

To make this trade-off, in this paper we use so-called system scenarios. These scenarios bundle similar control paths, while still allowing sufficient optimizations. The problem treated in this paper is: what are the right scenarios? i.e., which paths should be grouped together. For complex CFGs the number of possible scenarios (ways of grouping CFG paths) is huge; it grows exponentially with the number of CFG paths. Therefore heuristics are needed to quickly discover reasonable groupings. The main contribution of this paper is that we propose and evaluate three of these heuristics on both synthetic benchmarks and on a real-life application.

I. INTRODUCTION

Modern multimedia systems are characterized as applications with huge amount of data transfers and large memories. The memory subsystem in these applications consumes a major part of the overall area and energy. The optimization of global memory accesses in the memory subsystem usually brings significant energy savings [3]. Improving the locality of the memory accesses also decreases the life-times and hence the required memory footprint. Loop transformations like interchange and fusion are essential to achieve this goal.

Locality improving loop transformations are nearly always performed on a geometrical model [13]. This model can deal only with the static control parts (SCoPs) [2] of the code. The SCoP is a maximal set of consecutive statements without while loops, where loop bounds and conditionals may only depend on invariants within this set of statements. These invariants include symbolic constants, formal function parameters and surrounding loop counters. Intuitively, we can look at the SCoP as the geometrical model “basic block” on which the transformations can be applied. Note that the SCoP has a much coarser granularity than the traditional (compiler based) basic block. It contains usually non-perfectly nested for loops with affine and manifest bounds and affine and manifest if’s.

Modern multimedia systems can have quite complex control flow graphs (CFGs) with a lot of data-dependent conditions which create the boundaries of the SCoPs. The rich data-dependent control-flow limits the size of the SCoPs and so prohibits global loop transformations. To go beyond the scope of the SCoP, and enable far more global optimizations, we could consider paths of the CFG in separation. However, coding all paths separately would cause a tremendous code copying. In practice we have to make a trade-off between the extra optimization opportunities and the code size. To make this trade-off, in this paper we apply the system scenario technique [12]. This technique bundles similar control paths, while still allowing sufficient optimizations.

The systematic scenario concept was first used in [14] to capture the data dependent dynamic behavior inside a thread, in order to better schedule a multi-threaded application on a heterogeneous multi-processor architecture. In [8] scenarios have been used to refine the estimation of the Worst Case Execution Time (WCET). In [4], the authors propose a mapping technique and compiler which identifies the hot path(s) and merges or replicates the computational kernels, called Packet Processing Functions (PPF), in order to maximize system throughput. In [9] authors try to formalize the steps of a scenario methodology that can be applied in different contexts. None of these works focus on heuristics to improve scalability.

The superblock scheduling [11] and trace scheduling [6] techniques for VLIW compilers have a similar goal as scenarios, i.e., to enlarge the exploration space for optimizations. By creating larger blocks, they allow global instruction scheduling and thus allow better utilization of architecture resources and improvement of the performance. However, the enlargement is performed at the basic block level which has much lower granularity than the SCoP level and thus it is not sufficient.

In general, the exploration space of the scenario creation technique is not scalable with the number of paths in the CFG of the application. Our objective in this paper is to provide several heuristics with different time requirements and quality which trade-off these two metrics. The rest of the paper is structured as follows. Section II briefly recap scenario creation technique and its benefits. Section III motivates the need for heuristics and proposes several scalable heuristics. In Section IV we evaluate these heuristics on a real-life test-vehicle and on a set of synthetic examples.

II. SCENARIO CREATION

The scenario technique [12] assumes applications with one time (while) loop processing the (video or audio) frames. Each frame is processed in several computational kernels. With
computational kernel we understand a related set of statements that describes the enclosed part of the functionality of the application, e.g., filtering, requantization, reordering, antialias, FFT, IMDCT etc. and meet the SCoPs requirements. The selection and order of computational kernels is dependent on the type of the input frame and is determined by the data-dependent control flow within the time (while) loop. Most of the modern real-life multimedia applications, i.e., different audio and video codecs like MP3 audio decoder, fulfill this description of the application. However, our technique is also suited for any application with one time (while) loop, processing the objects differently (in different modes) depending on the type of the object where the processing can be modeled as the directed acyclic CFG with SCoP nodes.

![diagram](image)

**Fig. 1.** Example of creating scenarios from the original CFG.

Figure 1 depicts an example of a CFG and its decomposition into individual paths. On the left side is the original CFG followed by the equal sign and the 6 individual paths occurring in this graph split by plus sign. The individual paths enable far more global optimizations. However, to optimize each path separately will lead to a code explosion. Hence, some paths have to be grouped together. This is depicted by the boxes surrounding the paths which will be grouped together. Thus, on the right side of the figure (after the equal sign) we have 3 groups for our simple example. The first group is labeled scenario 1 and contains one path. The second group is labeled scenario 2 and contains three paths. The third group is labeled scenario 3 and contains two paths. The paths we group together within one box create control-flow subgraphs (CFsGs). Thus the terms CFsG and scenario are equivalent in the following text. This grouping solution (set of CFsGs) is just an illustration randomly chosen here. The whole search space from which we can pick a solution contains \( B_n \) possibilities where \( n \) is the number of paths in the original CFG and \( B_n \) is the Bell number. The Bell number \( B_n \) is equal to the number of ways to partition a set (of all paths) into nonempty subsets (of scenarios) if the set (of all paths) has \( n \) elements. For our example in Figure 1 with \( n=6 \) the \( B_6 = 203 \).

Figure 2 depicts all 203 possibilities for \( n=6 \) using the MP3 audio decoder [10] in a 2D exploration space, Estimated data memory size vs. Nr. of Abstract Syntax Tree (AST) nodes in the application. In this space, the trade-off between global data memory optimizations and code copying is represented by the Pareto curve. The MP3 has a complex CFG with 26 nodes and 234 possible paths through the CFG. After detailed profiling [1], we identified that for all of the MP3 input streams, only 6 out of the 234 possible paths are frequently used in the decoding. We call these active paths and the scenario technique has focused on only those to group.

**III. HEURISTICS TO REDUCE THE SCENARIO SEARCH SPACE**

The number of possible groupings to evaluate, like the one in Figure 1, grows exponentially with the number of paths in the CFG. E.g. to evaluate all possible groupings of 6 paths takes \(~25\)s on Pentium IV. To evaluate all possible groupings of 7 paths takes already \(~100\)s and to evaluate all possible groupings of 14 paths is not feasible due to limited memory resources. Thus the brute force evaluation of the scenario creation technique is clearly not scalable w.r.t. the number of paths in the CFG.

**A. Coverage criteria heuristic**

When grouping paths together also other paths can occur in the resulting CFsG additionally to the paths the CFsG is composed of. In Figure 1 scenario 2 consists of 3 paths. Nevertheless, it contains also the path of scenario 1 because all graph edges that are in scenario 1 are also in scenario 2. Thus scenario 1 is fully covered by scenario 2, i.e., the whole functionality of scenario 1 can be found in scenario 2. The CFsGs set, where a scenario is fully covered by another scenario from the same CFsGs set is pruned from the exploration space because it contains two times the same functionality for scenario 1. Thus, because of full coverage of scenario 1 in scenario 2 the whole clustering in Figure 1 is skipped. Because we check the coverage of the scenarios we call this heuristic coverage criteria heuristic.

The solution in Figure 1 can still be a Pareto point if the optimization potential for scenario 1 is large. Our heuristic has pruned this Pareto point out of the exploration space. This is often not a problem because we usually do not prune another solution that is close in the exploration space and it is also a Pareto point. E.g., in Figure 1 splitting the scenario 2 in two sub-scenarios, one containing the path which contains node 2 and the other one containing the two remaining paths (which contain node 3), creates a solution that is kept in the exploration space because then any scenario is not covered by another scenario. Because of the dominance of optimization
potential in scenario 1 this new solution has similar quality to
the original solution in Figure 1.

B. Loss/Similarity (L/S) heuristic

The previous heuristic was based on generating all possible
CFSG sets and pruning them before evaluation. The sequel L/S
heuristic and Fruchterman-Reingold (F-R) heuristic construct
"optimal" CFSG sets directly. Our target is to construct a CFSG
set where similar paths in terms of the common nodes in the
paths are in one scenario and where also paths do keep
their big SCoPs when grouped. Note, that when grouping the
paths together, the SCoPs get smaller and we are interested in
this difference. The reasons are the following: when grouping
similar paths, the number of duplicated nodes will be much
smaller than when grouping paths that are different. Also, we
would like to group paths together which do not lose anything
from the loop transformation potential when grouped, i.e., after
grouping the big SCoPs do not get smaller.

We measure the similarity of two paths \( p_i \) and \( p_j \) as a ratio
of nodes common to both paths, and all nodes of those two
paths. To count all nodes of two paths we count the nodes in
the graph constructed by grouping those two paths. To count
nodes common to both paths we use the inclusion-exclusion
principle, i.e., the cardinality of the intersection of two sets is
equal to the sum of cardinalities of those two sets minus the
 cardinality of the union of the two sets

\[
\text{Similarity}(p_i, p_j) = \frac{\|p_i \cap p_j\|}{\|p_i\| + \|p_j\| - \|p_i \cup p_j\|}
\]

The Similarity is a value in the interval \([0, 1]\), 0 means that
the two paths are completely disjunct, 1 means that they are
equal. Note, that we count the AST nodes \( N_{\text{AST}} \), not the
CFG (SCoP) nodes, thus

\[
\text{Similarity}(p_i, p_j) = \frac{N_{\text{AST}_p_i} + N_{\text{AST}_{p_j}} - N_{\text{AST}_{p_i \cup p_j}}}{N_{\text{AST}_{p_i \cup p_j}}}
\]

We define the loss between two paths as loss of the loop
transformation potential when these two paths are grouped
together. Then the SCoPs are not so large as before and this
decreases the optimization potential. We compute this loss as

\[
\text{Loss}(p_i, p_j) = f_{p_i} \times \sum_{\text{Dep}_k \in p_i} V_{\text{Dep}_k}(p_i) + f_{p_j} \times \sum_{\text{Dep}_k \in p_j} V_{\text{Dep}_k}(p_j) - \left( f_{p_i} \times \sum_{\text{Dep}_k \in p_i \cup p_j} V_{\text{Dep}_k}(p_i) + f_{p_j} \times \sum_{\text{Dep}_k \in p_i \cup p_j} V_{\text{Dep}_k}(p_j) \right)
\]

where \( V_{\text{Dep}_k}(p_i) \) is the dependency size of the \( k \)-th dependency
along the path \( p_i \) in the CFSG \( \{p_i\} \) (resp. \( \{p_i \cup p_j\} \)
in the second line). The dependency starts and ends at different
nodes. It should cross only ones with input and output degree
one in the CFSG when going from the start node to the end
node of the dependency. The start node should have output
degree one and the end node should have input degree one.
Similar definition is valid for \( V_{\text{Dep}_k}(p_j) \). The \( f \) is the frequency
of the path or of the union of two paths.

After obtaining those two measures we compute the ratio
L/S for each pair of paths. This ratio defines the “distance”
between the paths. Note that the similarity is unit-less metric
and the loss not. We consider the similarity as relative adap-
tation of the loss, i.e., if the similarity is 1, the loss is not
increased. However, if the similarity is smaller, this penalty is
transferred to the loss. With this approach we obtained better
results as when using two unit-less metrics. We believe, it is
because the larger importance of the loss in our technique.
Then we sort the pairs of paths according to distance and
start to group paths with the closest distance. The closer the
distance, the larger similarity and/or smaller loss. An example
of such an approach is in Figure 3 where the edge labels
show the distance between the two paths (nodes). We start
from the grouping when each path is separate. Then we group
two paths with closest distance, i.e., 0 and 1. That is second
grouping possibility. The third grouping possibility we obtain
when grouping also paths 2 and 3 together. The next possibility
is to add path 4 to the group of 0 and 1, etc.

C. Heuristic based on Fruchterman-Reingold (F-R) layout

This heuristic is an extension of the previous one. Here, the
inverse of the L/S ratio defines an attractive force between
two paths (not the distance). Then we define also the repulsive
force that is equal to the multiplication of loop transformation
potentials of the two paths, i.e.,

\[
\text{Attractive force}(p_i, p_j) = \frac{\text{Similarity}(p_i, p_j)}{\text{Loss}(p_i, p_j)}
\]

\[
\text{Repulsive force}(p_i, p_j) = \prod_{p \in \{p_i, p_j\}} \left( f_p \times \sum_{\text{Dep}_k \in p} V_{\text{Dep}_k}(p) \right)
\]

(see Figure 3). We start the process
with random layout and definition of attractive and repulsive
forces. Then we apply the forces under the defined cool down
function [7]. We have used the default linear cooling function,
where cooling schedule begins with some initial temperature
(100 in our case) and gradually (linearly) in time reduces the
temperature to zero. The result is a new layout that is used
as a start for our grouping technique, as shown in Figure 3.
This approach is a bit slower compared to the previous one
because of the new layout computation.
The quality of the three heuristics will be shown on the real-life MP3 audio decoder and on a set of synthesized examples generated with an adapted TGF pseudo-random graph generator [5]. As we mentioned above, in the MP3 only 6 paths out of 234 paths are active, i.e., have non-zero frequency for our test bitstreams. In general, we will only take the limited set of paths that are most frequently active. Only those are the target of our scenario exploration. The remaining ones are grouped together to one scenario called backup scenario. The profiling we used for active path identification is context sensitive, i.e., our test streams have been music test streams for which this is a representative distribution of paths. In a different context, e.g., for speech decoding other paths will pop up as important and some important paths from the music context will disappear. This will lead to different scenario creation and it is the reason why our scenario technique is context sensitive.

For the synthetic examples, we use three sets: a small set, a middle set and a large set. We define the small/middle/large w.r.t. the complexity and size of the graphs (number of CFG nodes and number of active paths) generated for the set, not w.r.t. the number of graphs in the set. The information about the sets can be found in the Table I after each set in parentheses in the form (average CFG nodes in a graph, average active paths in a graph, nr. of graphs in the set). We compare the curves obtained with the proposed heuristics to the Pareto curve of the full exploration space (see Figure 2 for the MP3 audio decoder). We define the accuracy of the heuristic as the ratio between the areas below the two curves (the brute force $bf$ and the heuristic $heur$). Our assumption is that we integrate (compute the area under the curve) starting from the solution where each scenario is an individual path (leftmost point) and ending by the solution where all the paths create one scenario (rightmost point). The resulting accuracy ratio can be in the interval $(0, 1)$. The closer the ratio is to 1 the better is the heuristic. Thus Accuracy($bf$, $heur$) = $\int bf / \int heur$.

The results for the real-life MP3 audio decoder and for the artificial examples are in Table I. In the first column is the name of the testbench, in the second column are the heuristics applied for that testbench, the third column depicts how many solutions has been pruned from the whole exploration space, the forth and the fifth column depict accuracy of the heuristic and time required. In general, we can conclude that the coverage criteria is a slow but accurate heuristic. From the two fast heuristics, the L/S is the better one. The F-R heuristic has not very good accuracy. We believe, the reason is the strong repulsive force that is also partly involved in the definition of the attractive force (the loss) and should be better calibrated. The fast heuristics (L/S and F-R) provide only one solution for given number of scenarios (CTsGS). After analyzing the Pareto points obtained by the brute force, we have seen that several Pareto points can be obtained for given number of scenarios. We believe, taking more than the best heuristic solution for given number of scenarios for fast heuristics will result into additional improvement in those heuristics. However, the time requirement will then also increase.

The large set of synthetic examples deserves special attention. Here, it was not possible to obtain the results for brute force and coverage criteria heuristic due to limited memory resources (out of memory after 1 day of computing). The L/S and F-R heuristics are the only option we have for large graphs with a lot of paths. The large synthetic set shows the importance of the applied fast heuristics. If not using those, we would not be able to obtain any results for the large set. Due to the fact we could not obtain brute force results also the accuracy would not be available for the large synthetic set.

### TABLE I

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Heuristic</th>
<th>Pruned [%]</th>
<th>Accuracy</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP3 audio decoder</td>
<td>Brute force</td>
<td>0</td>
<td>1</td>
<td>375</td>
</tr>
<tr>
<td>(26,6,1)</td>
<td>Coverage criteria</td>
<td>80.0</td>
<td>0.91</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>Loss/Similarity</td>
<td>97.0</td>
<td>0.93</td>
<td>23.3</td>
</tr>
<tr>
<td></td>
<td>F-R layout</td>
<td>97.0</td>
<td>0.85</td>
<td>23.2</td>
</tr>
<tr>
<td>Small synthetic examples (17.5,20)</td>
<td>Brute force</td>
<td>0</td>
<td>1</td>
<td>16.6</td>
</tr>
<tr>
<td></td>
<td>Coverage criteria</td>
<td>16.6</td>
<td>0.99</td>
<td>13.1</td>
</tr>
<tr>
<td></td>
<td>Loss/Similarity</td>
<td>90.4</td>
<td>0.93</td>
<td>2.05</td>
</tr>
<tr>
<td></td>
<td>F-R layout</td>
<td>90.4</td>
<td>0.87</td>
<td>2.01</td>
</tr>
<tr>
<td>Middle synthetic examples (72.7,5)</td>
<td>Brute force</td>
<td>0</td>
<td>1</td>
<td>666</td>
</tr>
<tr>
<td></td>
<td>Coverage criteria</td>
<td>17.3</td>
<td>0.99</td>
<td>617</td>
</tr>
<tr>
<td></td>
<td>Loss/Similarity</td>
<td>99.2</td>
<td>0.90</td>
<td>3.56</td>
</tr>
<tr>
<td></td>
<td>F-R layout</td>
<td>99.2</td>
<td>0.83</td>
<td>3.59</td>
</tr>
<tr>
<td>Large synthetic examples (188,14,3)</td>
<td>Brute force</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Coverage criteria</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Loss/Similarity</td>
<td>99.9</td>
<td>NA</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>F-R layout</td>
<td>99.5</td>
<td>NA</td>
<td>99.1</td>
</tr>
</tbody>
</table>

**IV. RESULTS**

We conclude that the most accurate and the fastest heuristic are the L/S and the F-R, respectively. The F-R heuristic is much faster and more accurate than the brute force heuristic. The L/S heuristic has a very good accuracy and is also much faster than the brute force heuristic. However, the L/S heuristic is not very fast. The brute force heuristic is the most accurate but also the slowest. The Coverage criteria heuristic is the slowest and has the worst accuracy. The Loss/Similarity criterion is the fastest and has the best accuracy. The F-R criterion is the fastest and has a very good accuracy. The Coverage criteria criterion is the slowest and has the worst accuracy.

### REFERENCES