Automatic Specification Granularity Tuning for Design Space Exploration

Jiaxing Zhang, Gunar Schirner
Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA
Email: {jxzhang, schirner}@ece.neu.edu

***Abstract*— Algorithm Design Environments (ADE), such as Simulink, have been shown to be efficient for development, analysis, and evaluation of algorithms. Recent tools propose to facilitate algorithm / architecture co-design by bridging the gap from ADE to System-Level Design Environments (SLDE) through automatic synthesis from algorithm models to SLDL specifications. With the wide range of block characteristic (from simple logic functions to complex kernels) in the algorithm model, however, it is challenging to select a suitable compositional granularity for SLD Language (SLDL) blocks in the synthesized specification. A high volume of SLDL blocks of little computation will increase the number of mapping possibilities, whereas large blocks with heavy computation on the other hand allow inter-block fusion reducing the computational demands in the overall specification yet sacrificing the mapping flexibility.**

In this paper, we introduce an automatic specification granularity tuning mechanism to determine the granularity in the synthesized specification model hierarchy guided by the computational demands of algorithm blocks. Our granularity selection significantly simplifies the early design space exploration as only a meaningful block decomposition is exposed in the synthesized specification. It leads to an overall system with less computational demands by leveraging the block fusion capabilities in the ADE. At the same time our granularity decision ensures that sufficient flexibility remains in the system for exploring heterogeneous mapping of the algorithm. Our results on real world examples show that specification models can be synthesized with 80% efficiency through block fusion with 70-90% fewer but coarser grained blocks.

I. Introduction

With exponentially increasing chip capacities, the increasing heterogeneity, exploding functional demands while still meeting shrinking time to market, designing modern Multiprocessor System-On-Chip (MPSoC) has become a tremendous challenge. Top-down design methodologies are one solution to cope with design complexities. These methodologies start at a high abstraction level while keeping the system-level overview and incrementally define and realize the details of the overall solution [15]. Within the context of top-down methodologies, algorithm / architecture co-design that successively matches and adapts both algorithm and architecture to optimize power and performance offers great opportunities.

One promising approach to realize algorithm / architecture co-design combines the strengths of a dedicated Algorithm Design Environment (ADE) and a System-Level Design Environment (SLDE) [16]. ADEs, such as Simulink, offer extensive algorithm modeling capabilities supported by algorithm databases, comprehensive toolboxes and analysis capabilities [13]. SLDEs, such as PEACE/HOPE [5], Dedalus [1] and SCE [4], focus on system-level explorations through generated Transaction-level Models (TLM) investigating into allocation and configuration of processing elements and communication infrastructures; mapping of applications over these heterogeneous architectures and application scheduling. To bridge the gap between ADE and SLDE, *Specification Synthesis* automatically generates a system specification in a System-Level Design Language (SLDL), such as SystemC, out of the algorithm description captured in an ADE's model.

978-3-9815370-2-4/DATE14/©2014 EDAA

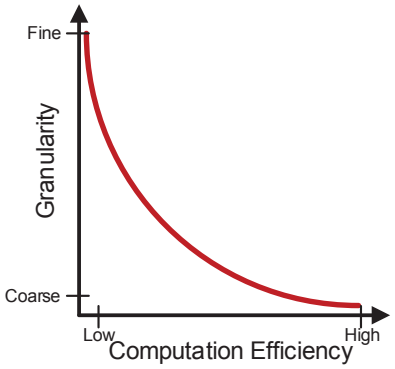


Fig. 1: Granularity and Efficiency Trade-off

The quality of the system specification is pivotal for the following system-level exploration. Important specification synthesis decisions include granularity of the SLDL specification, along with scheduling and exposed parallelism. These decisions impact the suitability of mapping computation to different processing element classes.

Of particular importance for the specification synthesis is the granularity selection. It defines the size (computational demands) for each SLDL block. When using an ADE model (i.e. Simulink) as an input, the ADE blocks may have a large variance in computational demands. It can range from blocks of simple logical operations with negligible computation to blocks containing a complex kernel, such as edge detection algorithms, with huge computational demands. A trade-off exists when selecting the specification granularity between number of blocks in the specification (more mapping alternatives), and reduction in the overall computational demands (efficiency). Fig. 1 illustrates the trade-off.

One extreme, the finest possible granularity, preserves all ADE blocks in the SLDL. This leads to the highest number of SLDL blocks and the greatest mapping flexibility. However, with many trivial blocks that potentially only contribute negligibly to the overall specification execution time, this only leads to an unnecessary high degree of freedom. It is not beneficial to split blocks that barely computationally contributing to different Processing Elements (PEs). At the same time, the unnecessary fineness in the model granularity selection may lead to additional overhead in block-to-block communication depending on system-level synthesis. The other extreme is the coarsest granularity that merges all algorithm blocks into a single SLDL block. This maximizes inter-block optimization opportunities, e.g. leveraging the block reduction techniques [8]. When synthesizing ANSI-C code with Simulink Embedded Coder (SEC) [8] we have observed 5-12% reduction (see Section V) in the total computational demands in the synthesized specifications between the finest and the coarsest granularities. However, due to merging all computation into a single block,

the mapping alternatives are dramatically impeded and parallel explorations become impossible.

Hence, when synthesizing SLDL specifications out of algorithm models, one important challenge is the granularity selection that strikes a balance between mapping possibilities (i.e. number of SLDL blocks) and inter-block optimizations (i.e. block fusions). Manually defining a proper granularity is impractical due to model size (with hundreds of blocks) and complexity with deeply nested hierarchy levels. Therefore, automatic methods for specification granularity tuning are needed.

In this paper, we analyze the granularity and efficiency trade-off for a set of media applications. We propose an automatic approach that, as part of the specification synthesis, selects certain algorithm blocks for fusion and thus tunes the effective granularity of the specification. Our automated granularity tuning approach, Computation-Guided Granularity Tuning (CGGT), bases the merging decision on computational demands of algorithm blocks. With a selectable merging threshold, the user is empowered to navigate the granularity and efficiency trade-off.

Our experimental results on three real-world examples demonstrate that the CGGT-based synthesis produces specifications achieving 80% of the maximal possible computational demand reduction (with single block only), while retaining sufficient number of blocks for further design space explorations.

This paper is structured as following: Section II introduces relevant work. Section III examines the background of specification performance analysis. Section IV discusses the granularity decisions. Section V demonstrates the benefits on experimental studies. Section VI concludes the paper.

II. Related work

Intense research effort has been invested into system-level design usually from system specifications. However, the specifications still need to be authored manually and little attention was given to how to synthesize a specification automatically. Approaches appeared for bridging multiple abstraction layers together, such as using UML as part of SoC design methodologies [7], [9] and language conversion techniques that translate UML models into SystemC [14], [12]. A framework for co-simulation between MATLAB and UML under a co-design flow with application specific UML customizations was provided in [10]. However, these work mostly focused on structural conversions and yet behavior translation, with exception of state machines, was less explored and the issue of the trade-off between block volumes and model performance was not observed.

A top-down refinement flow from Simulink to SystemC TLM was presented in [6], [11]. Refinement and component exploration in their work happened in Simulink, requiring direct annotation, rewriting and modifications of Simulink models. Unlike their methodologies focusing on the top-down workflow, we focus on the issues and opportunities emerged in bridging the two environments together. A re-coding approach [3] was proposed to transform flat C code into a structured SLDL specification. The effort similarly aimed to close the specification gap, but directly targeted C as the input language. Our approach, on the other hand, starts from a higher abstraction by using Simulink as input models. Furthermore, their approach relied on a re-coding scheme by manually determining the granularity without inter-component optimization.

III. Background: Specification Performance Analysis

This section analyzes a video MJPEG Encoder to provide further background information and substantiate the motivation for this work. For this paper we chose Simulink as an ADE. The Encoder model in Simulink contains 94 leaf blocks (not sub-divided into further blocks) and 70 hierarchical blocks with a nested hierarchical depth of 12. We analyze each leaf block in the Simulink model for its computational demands by defining it as the number of operations that a block performs in the total execution period. For simplicity, we define $BlockSize$ as the computational demands of a block.

To gain insight into the model composition, Fig. 2 plots a histogram over the algorithm blocks regarding $BlockSize$. The logarithmically scaled x-axis sorts blocks into bins by $BlockSize$. For each bin, it shows the relative number of blocks in yellow (left), and the relative contribution to total execution time in blue (right). Fig. 2 shows that more than 83% of all leaf blocks have $BlockSize$ less than 1 Million operations performing only simple and individual operations. While largely contributing to the total number of blocks, these do not contribute to the total computational cost. Only a few blocks (less than 17%) are of 1 Million operations or larger in size. In our model, these blocks include kernel computations such as template matching and DCT (Discrete Cosine Transform) for processing arrays of pixels. Often, computationally intense blocks are predefined library blocks in Simulink with un-explorable internal structures as encrypted MATLAB code or binary distributions. In this model, essentially only the large blocks contribute to the overall computational demands.

The Pareto principle (i.e. 80-20 rule) is clearly observable in the MJPEG Encoder analysis; where about the 20% of blocks contribute to 80% of the total computational load. At the same time we observe a strong drop in contribution to the total computation as the blocks get smaller in *BlockSize*. We therefore chose *BlockSize* as a metric to decide which blocks should be merged together during SLDL specification synthesis. Merging blocks controls the SLDL specification granularity and leverages the inter-block code optimization potential for synthesizing SLDL leaf blocks. This will result in a reduced overall computational demands of the generated SLDL specification while still offering sufficient number of blocks for heterogeneous DSE at system-level. At the same time this exposes only blocks with meaningful computational contributions, simplifying the DSE by reducing the total number of blocks to be mapped.

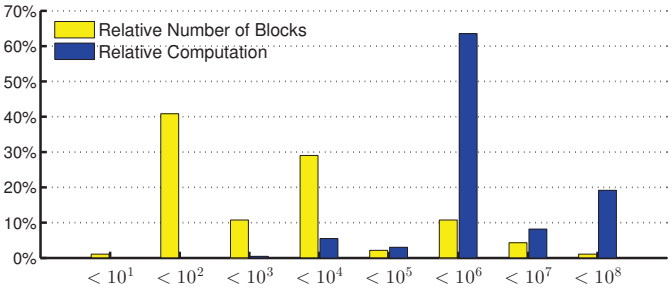


Fig. 2: MJPEG Encoder Algorithm Block Characteristics

IV. Specification Granularity Tuning

With the large number of blocks and high model complexities, automated methods are required for identifying algorithm blocks to be merged in the specification synthesis. Motivated by the strong correlation of $BlockSize$ to the total computational demands, we propose our Computation-Guided Granularity Tuning (CGGT) approach. This section first overviews the overall specification tuning / synthesis flow and then introduces the CGGT algorithm in detail.

A. Specification Granularity Tuning Flow

Fig. 3 illustrates the overall flow. The input to the flow is an algorithm model captured in Simulink. The algorithm model is then profiled to determine the computational demands ($BlockSize$) of each algorithm block. CGGT utilizes $BlockSize$ of each block in the model to determine the leaves for the SLDL specification. Finally, *Specification Synthesis* module realizes the granularity decision and generates accordingly the SLDL specification as an input to the system-level design space exploration.

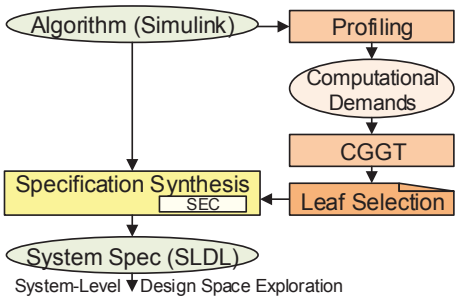


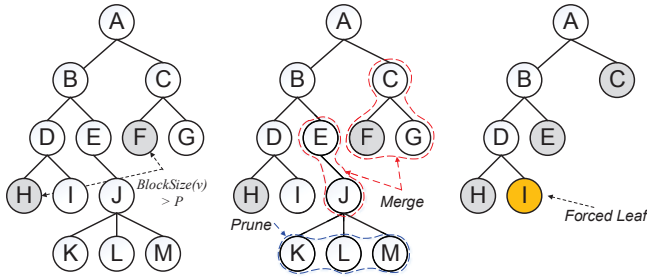
Fig. 3: CGGT Flow

Profiling: The $BlockSize$ of all blocks the algorithm model is estimated as a target-independent metric (e.g. add, subtract, multiply, divide) for gauging the computational impacts. The $BlockSize$ is not a measure of execution time. Nonetheless, it can be correlated to execution time taking into account target-specific processing capabilities and algorithm mapping of the specification.

For the purpose of this work, we chose the profiling phase of [2], which obtains basic block execution information through simulation and statically analyzes the code to compute the computational demands. For this, the algorithm model is synthesized into the SLDL specification at the original granularity and analyzed through the profiling phase of [2]. Other methods for determining the $BlockSize$, such as static analysis, are feasible, however do not change our overall proposed approach[1].

CGGT: Blocks in the algorithm model are distinguished between leaf and hierarchical blocks. In this step, CGGT identifies the suitable leaves (i.e. selecting hierarchical blocks to be leaf blocks) by comparing the $BlockSize$ against a computational lower bound threshold P and an upper bound threshold U. If the $BlockSize$ of a hierarchical block (e.g. sum of all children's demands) sits between these two bounds, it will be marked as a leaf for later synthesis. Details of the selection algorithm are discussed in Section IV-B. As its output, CGGT defines the *Leaf Selection* for *Specification Synthesis*.

[1]The Simulink integrated profiler currently only provides host execution time during simulation, which is insufficient for determining platform-independent processing demands.



(a) Input Model Tree (b) CGGT Process (c) Output Spec. Tree

Fig. 4: Graph Representation for CGGT Example

Specification Synthesis: Given the selection of leaves identified by CGGT, *Specification Synthesis* produces the SLDL specification converting the decision into reality. Each designated leaf is generated into ANSI-C code using the Simulink Embedded Coder (SEC). Inter-block optimizations are enabled to reduce the overall computational demands if a designated leaf contains multiple blocks. In addition, *Specification Synthesis* transposes the generated leaf ANSI-C code into SLDL and generates SLDL hierarchical blocks according to the algorithm model. Details of this process described in [16]. The resulting SLDL specification is input to a top-down system-level design space exploration flow to evaluate heterogeneous platform and specification mapping alternatives. The next section describes CGGT at the heart of our approach.

B. Computation-Guided Granularity Tuning

To more formally define CGGT, we define the following notations: an algorithm model hierarchy can be viewed as a spanning tree graph ($G(v, e)$) where a vertex (v) is a block and tree edges (e) represents containment relationship. The example in Fig. 4 shows a simple Simulink model structure with top-level block A containing B and C, which in turn contain other sub-blocks (e.g. $Child(A)$ returns B, C and $Parent(C)$ returns A). For simplicity of explanation, we assume unique block instances; hence blocks have a single parent. Only tree edges are allowed which describe hierarchical relationships and therefore back edges, forward edges or cross edges do not exist in our formulation. Each leaf block is annotated with the $BlockSize$ reflecting the computational demands obtained by the profiling – see Tabl. I for our example. The $BlockSize$ of a hierarchical block (i.e. a block with children) is defined as the sum of all its children.

To identify blocks based on the $BlockSize$, CGGT employs a lower bound threshold P. P denotes the minimal $BlockSize$ of a block to be considered as a leaf (i.e. $BlockSize(v) > P$). Otherwise, the leaf identification needs to propagate up the hierarchical tree to combine more blocks together to meet P. Meanwhile, the merging needs to be stopped with an upper bound threshold U which controls the largest tolerable $BlockSize$ of a block. Not using the threshold U would lead to a single root block as all the blocks will be

TABLE I: Leaf $BlockSize$ and Tuning Threshold

	H	I	K	L	M	F	G	P	U
$BlockSize$	100	40	20	30	50	120	5		
Threshold								90	130

merged. In essence, a block is declared as a leaf, if:

$$P < \sum_{\forall v \in Child(block)} BlockSize(v) < U \quad (1)$$

The CGGT algorithm (see Algorithm 1) builds upon the Depth-First Search (DFS) to cover all nodes in the graph (i.e. all blocks in the model) using a maker labeling approach for the leaf block selection. During the tree traversal by the algorithm it declares a node (*block*) as either leaf or hierarchical block by $Mark(block) \leftarrow LEAF \vee HIER$. To illustrate the operation, Fig. 4 together with Tabl. I shows an example. Prior to the algorithm running, all blocks in the model have no markers. CGGT thus marks all blocks that are without children (i.e. leaves in the algorithm model) as leaves (line 13) by recursively reaching to the deepest blocks. In our example, these are H, I, K, L, M, F, G.

Upon encountering a leaf block v with $BlockSize(v) > P$, such as H and F (highlighted in gray), it will mark its parent as $HIER$ (line 19). As H, F are leaves with sufficient computation, they label their parents as a hierarchical block ($HIER$) because CGGT works heuristically by separating blocks of big $BlockSize$ as much as possible within a parent block. By setting the $HIER$ marker, it signals that there are heavy blocks within such block. Thus, this parent block cannot be overwritten by the $LEAF$ maker anymore to preserve the earlier identified leaf blocks (H, F). On the other hand, these light leaf blocks are only able to claim their parents to be a leaf unless there is no other blocks have marked it as $HIER$. The $LEAF$ label of a block can be overwritten by other sibling blocks if any has a $BlockSize > P$ (line 14 - line 20).

Nevertheless, if none of the block's children has $BlockSize$ larger than P, the block (with clean makers) will be marked as a $LEAF$ (line 16). Essentially, v propagates the leaf to one level above based on computational compositions. The $BlockSize$ of the recent marked leaf block is again examined to decide if it needs to be further aggregated up in the MERGE procedure (line 1). In our example shown in Fig. 4(b), K, L, M are all pruned off by marking their parents to be a leaf, yet their parent J satisfies the threshold P (i.e. $BlockSize(K) + BlockSize(L) + BlockSize(M) > P$), which makes it a leaf that should be preserved in the hierarchy.

Once a hierarchical block is reached in the tree traversal, $MERGE$ procedure (line 2 - line 9) examines the $BlockSize$ of it against U to check if the block can be still considered as a leaf to further prune off undesired computational demands. The input block to $MERGE$ has a marker of either $LEAF$ or $HIER$ labeled by its child blocks. If the $BlockSize$ of this hierarchical block is less than U, it will be marked as a leaf and propagate up the leaf marker. Otherwise, it remains as $HIER$ and sets its own parent to be $HIER$. In our example, F, G are merged to their parents because the $BlockSize$ of C, which is the combined size of F and G, is less than U, and similarly J is merged because it is the only block, which apparently satisfies the requirement of U. The CGGT algorithm recursively walks through the entire model tree. Once the root node is visited, the leaf identification process completes. Fig. 4(c) shows the output specification tree. Most of the trivial blocks (K, L, M, G) are pruned off as their $BlockSize$ is smaller than P. The merging threshold U contributes further to trim off combinations of heavy and small blocks. J and F are merged to a level up without impeding their parent $BlockSize$ too much. However, I remains in the final specification as the $BlockSize$ of its parent block D is above the merging threshold (i.e. $U < BlockSize(D) = BlockSize(H) + BlockSize(I)$). Therefore, it stays as a leaf. This kind of leaf blocks are considered as *forced leaf* blocks. The number of forced leaf blocks is an important metric in order to evaluate the efficiency and performance of the tuning parameter selection. The goal is to minimize the number of forced leaves since they have small computation, thus are not desirable for later DSE.

Algorithm 1 Computation-Guided Granularity Tuning

```
1:  procedure MERGE(block, U)
2:      if BlockSize(block) < U then
3:          Mark(block)) ← LEAF
4:          if Mark(Parent(block)) ≠ HIER then
5:              Mark(Parent(block)) ← LEAF
6:          end if
7:      else
8:          Mark(Parent(block)) ← HIER
9:      end if
10: end procedure
11: procedure CGGT(block, P, U)
12:     if Child(block) = ∅ then
13:         Mark(block) ← LEAF
14:         if BlockSize(block) < P then
15:             if Mark(Parent(block)) ≠ HIER then
16:                 Mark(Parent(block)) ← LEAF
17:             end if
18:         else
19:             Mark(Parent(block)) ← HIER
20:         end if
21:     else
22:         for all v ∈ Child(block) do
23:             CGGT(v, P, U)
24:         end for
25:         MERGE(block, U)
26:     end if
27: end procedure
```

V. Experimental Results

To demonstrate the benefits of CGGT, we have implemented our approach and applied to real world examples. In this section, we first outline the realization of our approach and describe the experimental setup. We then introduce our examples which are three media applications: MJPEG Encoder, MJPEG CODEC and Corner Detection, and their characteristics. We will analyze CGGT on MJEPEG Encoder in detail and lastly generalize over all examples.

To validate the CGGT approach we have integrated it into our *Specification Synthesis* approach [16]. It uses a Simulink model as an input and generates an SLDL specification for further design space exploration. The *Specification Synthesis* relies on Simulink Embedded Coder (SEC), release 2011b, for generating the computation code (in ANSI-C) for the SLDL leaves. We have enabled the inter-block optimizations in SEC

TABLE II: Media Application Characteristics

Model Characteristics	Num. of Blocks	Num. of leaf Blocks	Comp. of original model (operations)	Comp. of all merged (operations)	Speed-up
MJPEG Encoder	140	93	83.8M	75.2M	10%
MJPEG CODEC	315	202	113.2M	99.1M	12%
Corner Detection	57	45	195.4M	184.3M	5%

to reduce computational demands for SLDL leaves containing multiple algorithm blocks. To obtain target-independent computational information, we integrated a C source code based profiler [2]. To demonstrate the efficiency of the algorithm, we have applied it to three real-world media examples: MJPEG Encoder, MJPEG CODEC and Harris Corner Detection. The examples stem from Simulink toolboxes as part of the 2011b release. For run-time validation, the MJPEG examples operate on a VGA size video stream of 900 frames, and the Corner Detection on a 1000 frame stream of the same size.

Tabl. II lists the model characteristics of our three media applications. MJPEG Encoder and CODEC consist of 140 and 315 blocks respectively. Out of these, 93 and 202 blocks are leaf blocks. The Corner Detection is smaller, containing only 57 blocks, among which 45 are leaves. Many of the core blocks of the Corner Detection are described in isolated units (S-Function or library modules) that express a large computational demands. With the limited visibility, these blocks cannot be decomposed into smaller blocks.

To obtain an indication of the optimization potential, we synthesized each model once as *original model* (i.e. all blocks are preserved in the SLDL specification), and once as *all merged* yielding the coarsest possible granularity (i.e. synthesizing one SLDL block representing the root of the algorithm). Tabl. II lists the total amount of computation for both extremes. The total computational demands drops by 10% and 12% for the MJPEG examples. However, the Corner Detection example profits less with only 5% since it starts out with coarser algorithm blocks. The following paragraphs analyze how much the optimization potential can be retained while merging computationally non-contributing blocks.

A. Granularity Tuning Evaluation

As described in the CGGT algorithm, the lower bound threshold P is used for leaf identification. As such, it impacts the resulting number of blocks and the inter-block optimization potential. In order to make our measurements independent of the example size, we seek an example-relative definition of P. For this, we propose the $BlockSize$ cumulative contribution percentile for P. For instance, $P = 80\%$ indicates the threshold has the value such that 80% of all blocks in an example model has equal or smaller $BlockSize$. In the MJPEG Encoder example, $P = 80\% = 1.1MOps$ (see Fig. 2). For simplicity, we define the upper bound merging threshold U as $U = 1.1P$. The overall effect of both parameters on a resulting specification can be observed while tuning P to a larger value yielding more coarse-grained model hierarchy. Fig. 5 visualizes the results when varying the threshold P from 0% to 100% at three intervals: 20%, 40% and 80% using the MJPEG Encoder example. The plot shows how leaf blocks with increasing $BlockSize$ (X-axis) contribute to the cumulative computation (Y-axis). Note that both axes are in log scale. The cumulative computation indicates that

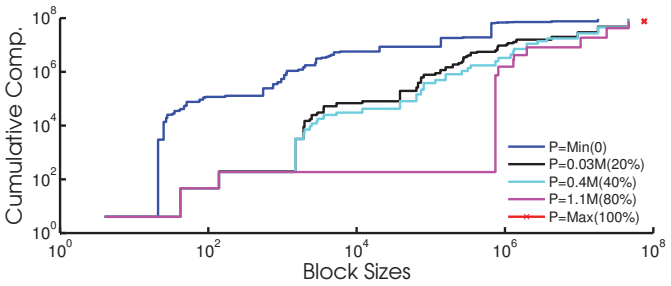


Fig. 5: MJPEG Block Reduction

blocks of a certain $BlockSize$ or smaller have contributed to the total computational demands. To give an example: in the original granularity (blue line), blocks with a $BlockSize$ of 100 operations or less attribute to total computational demands of 10000 operations.

Given in Tabl. III, we observed that tuning P from a lower to a higher threshold introduces a block removal associated with P. Both Fig. 5 and Tabl. III show that a higher P produces fewer total blocks with bigger $BlockSize$ as small blocks have been merged together. Due to the model hierarchical limitations, it is not possible to remove all trivial blocks while only keeping heavy blocks. The number of forced leaves shows this effect. By increasing P, the forced leaves are reduced from 17 blocks to 5 blocks as some of these trivial blocks exist at a higher hierarchical level and they cannot be merged without merging other extra heavy blocks together. The most effective block reduction is obtained with $P = 80\%$, which produces a specification with 12 leaf blocks and only 5 forced leaves. The total model speedup is around 8% comparing to the original model with full block specification synthesis. As previously highlighted in Fig. 2, blocks between 0.1M and 1M size contribute over 80% to the total computational demands. Only 20% of blocks have a $BlockSize$ larger than 1.1M, yet they constitute 80% of the total model computational cost.

TABLE III: Tuning Effect on MJPEG Encoder

Threshold (P)	Num. of Blocks	Num. of leaf Blocks	Num. of Forced Leaves	Model Comp. Demands	Speedup	Efficiency
$P = 0$	140	93	0	83.8M	0%	0%
$P = 20\%$	94	71	17	81.4M	2%	28%
$P = 40\%$	46	31	9	78.6M	6%	60%
$P = 80\%$	19	12	5	76.7M	8%	82%
$P = 100\%$	1	1	0	75.2M	10%	100%

In order to simplify the comparison across examples, we define efficiency as a relative metric. It shows how much of the best possible intra-block optimization (i.e. when all blocks are merged into one) is achieved by the model. For instance, in the MJPEG Encoder the optimization potential is $83.8M - 75.2M = 8.6M$. With P = 80%, the total computational demands is 76.7M. In absolute terms, 7.1M operations were avoided, yielding an efficiency of 82%. Tabl. III lists the efficiency in the last column. Fig. 6 illustrates the effect of increasing the threshold P in finer detail. It shows the threshold P on the X-axis. The left (blue) Y-axis indicates the resulting number of blocks, and the right (green) Y-axis the efficiency. The MJPEG Encoder starts out with many blocks and low efficiency. As P increases, number of SLDL blocks decreases and efficiency increases. Both correlations are almost linear.

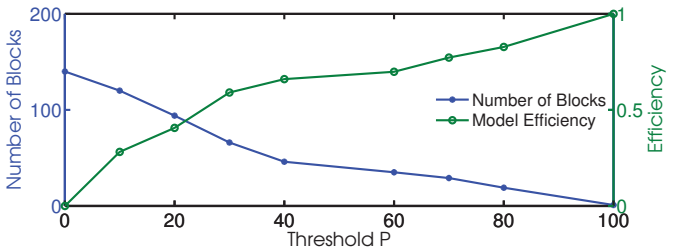


Fig. 6: MJPEG Granularity vs. Efficiency

B. Block Fusion Optimization and Mappablity Trade-off

In order to generalize our observations, we have expanded to include all three media applications. Fig. 7 illustrates the trade-off between number of blocks, which impact mapping flexibility, and efficiency (utilizing the inter-block optimizations). In order to normalize across all applications, the y-axis shows the number of blocks relative to the maximum number (i.e. original granularity). The x-axis denotes efficiency as defined earlier. Each application is shown as a line obtained by sweeping the threshold P at multiple intervals.

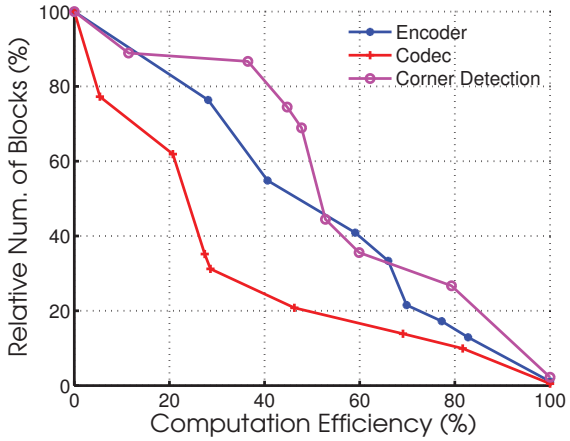


Fig. 7: Granularity vs. Efficiency

All three models converge to higher efficiency with a higher threshold P. The Corner Detection, with its predominantly large blocks, increases very little in efficiency with small P ($P < 50\%$), showing a limited block reduction. With larger P ($P > 50\%$), it follows the same trend as the other examples, with linearly increasing efficiency.

Fig. 7 numerically validates the trade-off introduced in our motivating discussion (see Section I). It serves a tremendous value for designers to determine the leaf selection for a synthesized specification. The CGGT approach enables designers to navigate the granularity / efficiency trade-off, and to balance the number of blocks for exploration with overall performance.

VI. Conclusion

In this paper we have identified the trade-off between model granularity and intra-block optimization in a specifications synthesis process. Model granularity impacts the specification mapping flexibility as well as efficiency. Many small blocks with negligible contribution to computation may only introduce overhead, may not lead to meaningful design space exploration alternatives, and thus only artificially complicates any DSE. Conversely, few large blocks reduce the overall computational demands through inter-block optimizations during the specification synthesis, but will limit the mapping flexibility. In this paper, we have analyzed and quantified the trade-off through three real-world media examples.

To aid the designer in navigating the trade-off, we introduced Computation-Guided Granularity Tuning (CGGT). It identifies leaf blocks with block fusion to produce an overall specification model in desired granularity as specified by the user. Our approach enables designers to navigate the trade-off to produce specifications with up to 80% computational performance efficiency with 70-90% block reduction. Our future work will enhance the metric for leaf selection to include communication, and we will analyze the impact of the mapping flexibility with heterogeneous platforms mapping.

Acknowledgment

The work presented in this paper is partially supported by the National Science Foundation under Grant No. 1136027.

References

[1] M. Bamakhrama, J. Zhai, H. Nikolov, and T. Stefanov. A methodology for automated design of hard-real-time embedded streaming systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 941–946, 2012.

[2] L. Cai, A. Gerstlauer, and D. D. Gajski. Retargetable profiling for rapid, early system-level design space exploration. In *Proceedings of the Design Automation Conference (DAC)*, San Diego, CA, June 2004.

[3] P. Chandraiah and R. Dömer. Code and data structure partitioning for parallel and flexible MPSoC specification using designer-controlled recoding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1078 –1090, June 2008.

[4] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski. System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design. 2008(647953):13, 2008.

[5] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y. pyo Joo. PeaCE: A hardware-software codesign environment for multimedia embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 12, 2007.

[6] A. A. Jerraya, A. Bouchhima, and F. Ptrot. Programming models and HW-SW interfaces abstraction for multi-processor SoC. In *Design Automation Conference*, pages 280–285, 2006.

[7] G. Martin and W. Mueller. *UML for SOC Design*. Springer, 2005.

[8] The MathWorks, Inc. *Simulink Embedded Coder Ref. R2011b*, 2011.

[9] S. J. Mellor and M. J. Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley Professional, May 2002.

[10] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, and Y. Vanderperren. UML for ESL design: basic principles, tools, and applications. In *International Conference on Computer Aided Design*, page 7380, 2006.

[11] K. M. Popovici. *Multilevel Programming Envrionment for Heterogeneous MPSoC Architectures*. PhD thesis, Grenoble Institute of Technology, 2008.

[12] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A SoC design methodology involving a UML 2.0 profile for SystemC. In *Design, Automation, and Test in Europe*, pages 704–709, 2005.

[13] The MathWorks Inc. MATLAB and Simulink, 1993-2013.

[14] Y. Vanderperren and W. Dehaene. From UML/SysML to MATLAB/Simulink: current state and future perspectives. In *Design, Automation, and Test in Europe*, 2006.

[15] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor System-on-Chip (MPSoC) Technology. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 27:1701–1713, 2008.

[16] J. Zhang and G. Schirner. Joint algorithm developing and system-level design: Case study on video encoding. In *Embedded Systems: Design, Analysis and Verification*, pages 26–38. Springer, 2013.