A Synthesis-Parameter Tuning System for Autonomous Design-Space Exploration

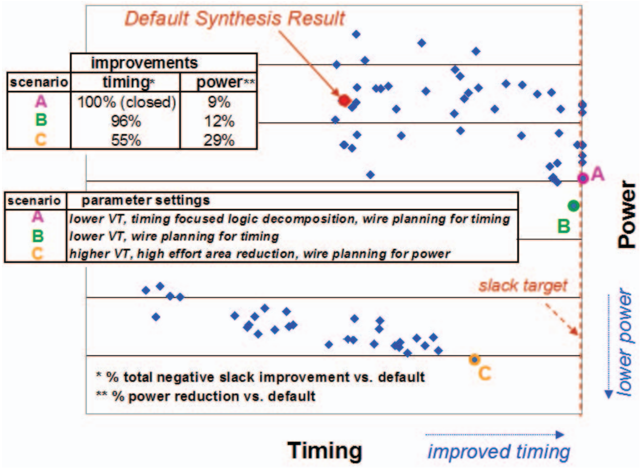
Matthew M. Ziegler[1], Hung-Yi Liu[2*], George Gristede[1], Bruce Owens[3], Ricardo Nigaglioni[4], and Luca P. Carloni[2]
[1] IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
[2] Department of Computer Science, Columbia University, NY, USA
[3] IBM Systems & Technology Group, Rochester, MN, USA
[4] IBM Systems & Technology Group, Austin, TX, USA

***Abstract*— Advanced logic and physical synthesis tools provide a vast number of tunable parameters that can significantly impact physical design quality, but the complexity of the parameter design space requires intelligent search algorithms. To fully utilize the optimization potential of these tools, we propose SynTunSys, a system that adds a new level of abstraction between designers and design tools for managing the design space exploration process. SynTunSys takes control of the synthesis-parameter tuning process, i.e., job submission, results analysis, and next-step decision making, by automating a key portion of a human designer's decision process. We present the overall organization of SynTunSys, describe its main components, and provide results from employing it for the design of an industrial chip, the IBM z13 22nm high-performance server chip. During this major design, SynTunSys provided significant savings in human design effort and achieved a quality of results beyond what human designers alone could achieve, yielding on average a 36% improvement in total negative slack and a 7% power reduction.**

I. Introduction

Modern VLSI design is a quest to optimally tune and balance multiple objectives, such as power, performance, and reliability. The complexity of VLSI chips also continues to rise despite time-to-market pressures and the desire for cost savings via smaller design teams. These challenges are causing an industry shift from custom design techniques towards synthesis-centric methodologies for optimizing design quality and boosting design productivity. The complexity of modern designs coupled with multiple design objectives leads to the rise of difficult optimization problems beyond what even experienced designers can manage manually. Automated design space exploration (DSE) approaches, however, are well suited to handle many of these problems. In this paper we present a new system that automates the DSE process for the logic and physical synthesis steps within an industrial VLSI design flow. This system was successfully used during the design of the IBM z13 processor chip and was crucial for achieving timing closure as well as meeting power targets. The processor chip was designed in a 22nm SOI technology and has a clock frequency of approximately 5GHz. This high-performance server chip is the heart of the IBM z13 mainframe system [1].

Synthesis tools continue to become more sophisticated and provide numerous parameters to improve design quality. As an example of the wide design space available from modifying synthesis parameters, Fig. 1 shows the scatter plot of achievable design points for a portion of a synthesized floating-point multiplier macro. A macro may span from 1K to 1M gates in our context. Each point denotes the timing and power values achieved simply by tuning the input parameters of the synthesis program. The ultimate goal of this process is to reach timing closure at the lowest achievable power. Quite often the default values for the parameters are not ideal for a specific macro, which would benefit instead from parameter customization. Fig. 1 also highlights three scenarios (A, B, and C) along the Pareto frontier. These scenarios show the available tradeoffs between timing closure and power reduction, e.g., point A closes timing with a 9% power reduction, whereas point C improves timing by 55% with a 29% power reduction. These points along the Pareto set provide a number of potential steps towards the ultimate goal, depending on the additional techniques at the designer's disposal beyond parameter tuning. This example of a relatively simple macro underscores how significantly the parameters settings can affect a design.



	improvements	
scenario	timing*	power**
A	100% (closed)	9%
B	96%	12%
C	55%	29%

scenario	parameter settings
A	*lower VT, timing focused logic decomposition, wire planning for timing*
B	*lower VT, wire planning for timing*
C	*higher VT, high effort area reduction, wire planning for power*

Fig. 1. An example of the available design space by modifying synthesis parameters.

The high flexibility and sophistication of advanced synthesis tools increases their complexity and makes navigating the design space difficult, and sometimes non-intuitive for their users. Since the number of parameters for synthesis tools can be in the thousands and synthesis runs may take several hours or even days, exhaustive parameter-tuning is typically infeasible. Furthermore, while Fig. 1 portrays the design space of two metrics, designers often need to consider many more metrics, sometimes dozens, when evaluating the quality of a synthesis scenario. In summary, efficient DSE using advanced synthesis tools is increasingly challenging for even experienced professional designers and daunting for novice designers.

To efficiently and fully utilize the optimization potential of advanced synthesis tools, we propose a new system for managing the design space exploration process that we call the Synthesis Tuning System (SynTunSys). Our system amounts to a new level of abstraction where the human designer offloads the DSE task to automated tools. SynTunSys has been fully implemented and

[*] Hung-Yi Liu is now with the Intel Design Technology & Solutions Group, Hillsboro, OR, USA.

978-3-9815370-7-9/DATE16/©2016 EDAA

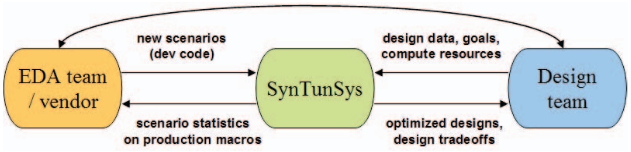

EDA team / vendor
new scenarios (dev code)
scenario statistics on production macros
SynTunSys
design data, goals, compute resources
optimized designs, design tradeoffs
Design team

Fig. 2. Design team, EDA team, and SynTunSys interaction.

deployed during the design of the IBM z13 22nm server processor. A systematic study of ~200 macros during the actual processor design cycle shows that SynTunSys provides, on average, 36% improvement in total negative slack (TNS) and 7% power reduction. Further, it improves the macro internal negative slack by 60%, on average.

II. Related Work

DSE solutions have been presented across various levels of abstraction related to integrated circuit and computer design. At the architectural level, many DSE studies have used models or simulators to explore multi-objective design spaces, e.g., [2]. These studies however, typically do not result in implemented designs. A number of high-level synthesis (HLS) studies have also employed DSE techniques, e.g., [3, 4]. In addition, DSE for FPGA synthesis using genetic algorithms was recently reported [5]. However, we know of no publications targeting fully automated DSE for logic and physical synthesis, which is our level of focus. Including physical synthesis in the DSE process achieves design optimization and completes a key step in the physical design process.

In contrast to prior works, there are many unique aspects of our DSE system. SynTunSys operates on general purpose synthesis tools, providing a solution for all synthesizable digital designs. Furthermore, SynTunSys is proven in an industrial setting, where it was essential to meeting performance, power, and schedule goals for a leading server processor currently in production. Thus, to our knowledge, SynTunSys is the first general, self-evolving, and autonomous parameter-tuning system for the logic and physical synthesis levels of design.

III. System Architecture

In order to perform automatic tuning of synthesis parameters, SynTunSys constructs synthesis scenarios (synthesis parameter settings), runs the synthesis jobs, analyzes the results, and iteratively refines the solutions. This decision loop, automated by SynTunSys, is a key component of the design process that is typically performed by human designers. The system employs parallel and iterative black-box search techniques to explore the design space in a manner that can efficiently scale to use the available resources in a compute cluster.

In addition to macro optimization, a secondary use model of SynTunSys provides a vehicle for in-house CAD teams or EDA vendors to deploy and test new synthesis optimizations on production macros, as Fig. 2 illustrates. SynTunSys is inherently tolerant of failures of individual synthesis jobs with little risk of degrading overall quality of results. This failure tolerance allows a small number of experimental scenarios to be run as part of DSE, enabling a DevOps deployment model. New experimental scenarios can be either provided to designers by the EDA team or run directly by the EDA team. SynTunSys collects synthesis-run statistics and keeps a history of results for all designs that can be monitored to determine a scenario's impact on design quality as well as trends across multiple scenarios and designs.

Fig. 3 shows the SynTunSys architecture. Although the architecture of the system is general and could be applied to earlier steps in a flow (e.g. HLS) or later steps (e.g. routing), we focus on the logical and physical synthesis steps because in our experience they offer a desired balance between (i) return on investment (ROI) of the compute resources and (ii) modeling accuracy of the final design implementation. SynTunSys consists of a main tuning loop that, at each iteration, involves running multiple synthesis scenarios in parallel, monitoring the jobs in flight, analyzing the results of the jobs, and making a decision for the next set of scenarios. A second background loop archives the results of all runs from all macros, users, and projects. The SynTunSys archive is a database that can be mined for historical trends across multiple macros and to provide feedback in terms of the performance of synthesis parameters.

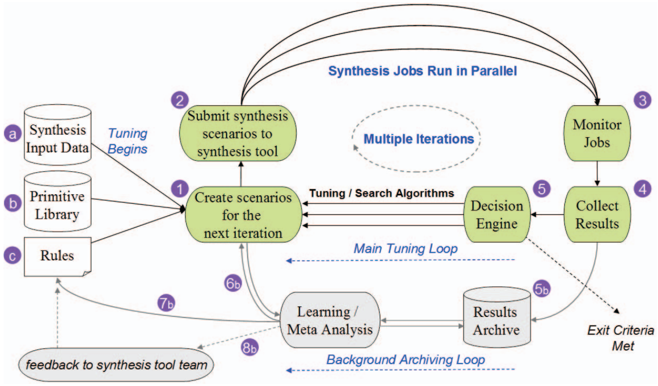


Fig. 3. Architecture of the SynTunSys process. The program employs a parallel and iterative tuning process to optimize macros.

The SynTunSys process begins at Step 1 in Fig. 3 where the initial synthesis scenarios are generated based on the following input data: (a) standard input data for circuit-level synthesis, which typically includes an RTL description, a physical abstract view providing macro boundaries, pin locations, and timing assertions; (b) a SynTunSys Rules file describing the primitives for the design space exploration and a cost function describing the optimization goals, among other options; and (c) a library of primitives that contains the detailed definitions of all potential exploration options. More details on (b) and (c) are provided later in this section. At Step 2 in the figure, multiple synthesis jobs are submitted in parallel to a compute cluster by issuing batch calls to the underlying synthesis tool. Following submission, SynTunSys starts a monitor process that tracks the synthesis jobs (Step 3). When either all jobs complete, or a large enough fraction of jobs complete, or a time limit is reached, the monitor process initiates the collection and analysis of the results of the parallel synthesis jobs (Step 4). Based on the collected results, a decision algorithm at Step 5 creates a new set of scenarios (synthesis parameter settings) to be run in the next iteration. These new jobs begin with the initial input data and are again run in parallel. The process iterates attempting to improve upon results until an exit criteria is met, e.g., the maximum number of user specific iterations is reached or the DSE algorithm experiences diminishing returns.

A. *Human Design Time vs. Compute Time*

In this work we make a key differentiation between compute time and human design time. SynTunSys significantly reduces human design time by offloading human designer effort to compute resources. Compute resources are scalable and more abundant over time, whereas there is a premium on training and retaining skilled designers.

The interaction between SynTunSys and a human designer can be viewed as a collaboration between human and computer, each with unique skill sets, rather than a conventional CAD point tool executing a fixed task. Freed from the parameter tuning task, designers can pursue parallel and complementary design techniques, e.g., design structuring [6]. SynTunSys results can also provide hints about the macro design space for skilled human designers to pursue further. The SynTunSys DSE results can also help educate novice designers. Furthermore, SynTunSys often leads to non-intuitive design points that even a skilled designer could not achieve alone.

Table 1. An example of primitive names and primitive descriptions.

Primitive Name	Primitive Description
restruct_a	Logic restructuring to reduce area
restruct_t	Logic restructuring to improve timing
rvt_lvt10	Native RVT, allow 10% LVT
rvt_lvt50	Native RVT, allow 50% LVT
vtr_he	High effort VT recovery
area_he	High effort area reduction
wireopt_t	Wire optimization for timing
wireopt_c	Wire optimization for congestion

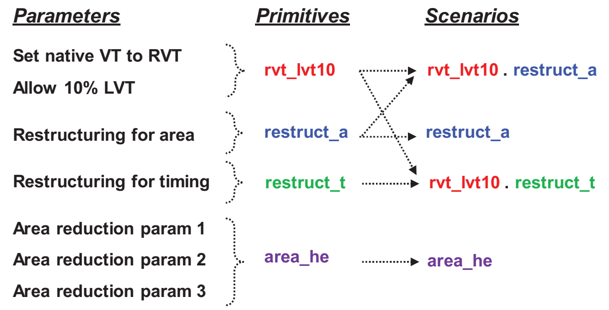
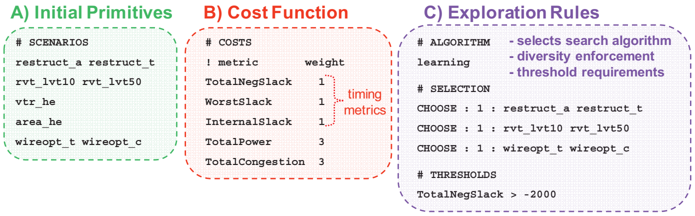


Fig. 4. Illustration of the interaction of parameters, primitives, and scenarios. Primitives consist of one or more parameters. Scenarios consist of one or more primitives.



```
# SCENARIOS
restruct_a restruct_t
rvt_lvt10 rvt_lvt50
vtr_he
area_he
wireopt_t wireopt_c
```

```
# COSTS
! metric        weight
TotalNegSlack     1
WorstSlack        1
InternalSlack     1
TotalPower        3
TotalCongestion   3
```

```
# ALGORITHM
learning

# SELECTION
CHOOSE : 1 : restruct_a restruct_t
CHOOSE : 1 : rvt_lvt10 rvt_lvt50
CHOOSE : 1 : wireopt_t wireopt_c

# THRESHOLDS
TotalNegSlack > -2000
```

Fig. 5. Components of the Rules file: A) the primitives to be explored during DSE; B) the cost function guiding the DSE; C) search algorithm, selection criteria, and configuration parameters.

The inherent SynTunSys runtime and disk-space overhead compared to a single synthesis job motivates streamlining the DSE search process with intelligent decision algorithms. Our current decision algorithms reduce the number of SynTunSys iterations to about 3-5 iterations, leading to a little over a 3-5x runtime (latency) increase versus a single synthesis run. Over the 3-5 iterations, approximately 100-200 scenarios are run. Although this overhead may seem costly, within the scope of a large design project it is quite tolerable and provides a high ROI for the following reasons: (i) SynTunSys is a fully autonomous system that does not require human designer effort once initiated. (ii) Running SynTunSys is not necessary every time a macro is synthesized; SynTunSys needs only to be run at certain points in the design cycle to locate customized parameters for a specific macro; during subsequent synthesis runs, the SynTunSys scenarios can be reused such that subsequent synthesis runs have no runtime overhead.

B. *Initial Design Space Reduction*

Parameters: Advanced synthesis tools can have a vast number of tunable parameters making an exhaustive design space search infeasible. In our specific case, the synthesis tool we employ has over 1000 parameters [7]. These parameters span the logic and physical synthesis space and the control settings for modifying the synthesis steps, such as: logic decomposition, technology mapping, placement, estimated wire optimization, power recovery, area recovery, and/or higher effort timing improvement. The parameters also vary in data type (Boolean, integer, floating point, and string). Considering that an exhaustive search of only 20 Boolean-type parameters leads to over one million combinations, it is clear that intelligent search strategies are required.

Primitives: To reduce the ~1000 multi-valued parameter space up front, we recast this DSE problem to have a space on the order of 100 Boolean parameters. This design space reduction involves a one-time offline effort to create a library of *primitives*. A primitive contains one or more synthesis parameters set to specific values. Table I shows an example of a small primitive library. The actual primitive library in our case consists of ~300 primitives. In general, a primitive targets a singular action. Thus, a SynTunSys primitive is a binary decision, whereas setting many parameters individually may require many more decisions.

Scenarios: SynTunSys creates scenarios consisting of one or more primitives that contain parameters, as Fig. 4 shows. The construction of scenarios is not trivial, motivating the need for intelligent decision algorithms. In particular, some primitives are complementary, e.g., “restruct_t” and “wireopt_t” for timing optimization, which however may require additional “wireopt_c” to compensate for routability. Other primitives are non-complementary, such as “restruct_t” and “area_he”; the former may upsize standard cells for larger driving strength or duplicate downstream cells for smaller capacitance load, while the latter may do the opposite changes for maximal area reduction. In practice, the optimal scenarios are macro-specific with respect to a set of weighted design objectives, while default settings require balancing QoR across multiple macros (see Fig. 1).

Remarks: The primitive library was chosen to speed up and focus the design-space search task, yet discretization of the design space via primitives may lead to unsearchable points. An alternative approach would be tuning individual parameters followed by feature extraction to determine attractive parameter settings. However, this fine-grained approach would have high computation cost and most likely would not yield strong results for quite some time.

To further reduce the design space to a reasonable size, independent of the size of the primitive library, a subset of primitives are selected from the primitive library for a specific SynTunSys run based on the optimization goals for the specific tuning run. This process of selecting a subset of primitives per SynTunSys run allows scaling the specific tuning run for the available compute resources.

C. *SynTunSys Components*

The SynTunSys Rules file contains configuration settings for a specific tuning run. Fig. 5 shows examples of the key sections of the Rules file: A) the primitives to be explored during the design space search; B) the cost function guiding the DSE; C) the overall search algorithm as well as the parameters to configure the search algorithm. These parameters can adjust the effort level of the search as well as the exit criteria, and are often modified depending on the available compute resources and/or the size of the macro.

Cost Function: The setting of the SysTunSys cost function conveys the optimization goals. It converts multiple design metrics into a single cost number, allowing cost ranking of scenarios. Examples of available metrics include: multiple timing metrics, power consumption, congestion metrics, area utilization, electrical violations, runtime, etc. The selected metrics are assigned weights to signify their relative importance. The overall cost function is then a “normalized weighted sum” of the *m* selected metrics, expressed by Equation (1) where *Norm(Mi)* is the normalized *Mi* across all the scenario results in a SynTunSys run.

$$Cost = \sum_{i=1}^{m} W_i \cdot Norm(M_i) \qquad \begin{array}{l} where: \\ W_i = weight_i \\ M_i = metric_i \end{array} \qquad (1)$$

Main Tuning Loop: The main tuning loop of SynTunSys begins when the monitor triggers completion, i.e., Step 3 in Fig. 3. This process includes a step that eliminates scenarios with erroneous or missing results data and also scenarios that fail to meet minimum design criteria, as specified by the designer. The resulting scenarios are all valid for cost analysis. After cost ranking, an optional routine may be applied to ensure there is diversity among the parameters driving the DSE, e.g., primitives having higher cost than competitors across a dimension of the primitive space may be removed. The decision engine is the final step of the main tuning loop.

Background Archiving Loop: The collected results of all runs are stored in a database, i.e., Step 5b in Fig. 3. The archived data provides a history of all previous runs that SynTunSys uses for multiple purposes. One use of the archived data is to generate DSE scenarios from historical primitive performance, represented by Arc 6b in Fig. 3. The example Rules files are also periodically updated based on the analysis of historical primitive performance (Arc 7b). The historical performance is also useful feedback to the synthesis tool development team (Arc 8b).

D. Descision Engine Algorithms

The SynTunSys decision engine algorithms are key components of the system that determine which scenarios should be run at each iteration. The decision engine is also a component that can be upgraded independently and we constantly look to improve these algorithms in terms of QoR prediction accuracy and compute efficiency. The general problem the decision engine addresses can be described as black-box tuning or search, i.e., we treat the underlying synthesis tool as black-box software during the parameter tuning process. The black-box tuning problem appears in numerous applications and has also been approached using a number of techniques, e.g., machine learning [8], Markov decision processes [9] and Bayesian optimization [10].

During the continued development of SynTunSys we have explored algorithms ranging from pseudo-genetic algorithms to adaptive learning. Due to space limitations, however, a detailed treatment of these algorithms will be deferred to a future publication.

IV. SynTunSys Results

SynTunSys was used during the design of the IBM z13 22nm server processor. The processor underwent two chip releases (tapeouts) over a multi-year design cycle, during which SynTunSys was applied to macros over both releases. The chip consists of a few hundred macros that average around 30K gates in size, with larger macros in the 300K gate range. Prior IBM server processors have also used systematic parameter tuning on a smaller scale. The IBM POWER7+ [11] and POWER8 [6] processors employed an earlier version of SynTunSys during the second chip releases, but mainly for power reduction purposes. In the case of the z13 processor, however, SynTunSys was employed during the entire design cycle for timing closure, power reduction, and improving macro routability.

During the first chip release, a dedicated SynTunSys team performed tuning for hundreds of macros across the chip. In parallel, a number of designers further tuned the macros they owned, as needed. Based on the efforts of the dedicated tuning team we were able to track SynTunSys results on approximately 200 macros from the processor core. The "pass 1" rows of Table 2 show the average improvements achieved by SynTunSys over the best solution previously achieved by the macro owners for the first chip release.

Note that these results are based on the routed macro timing and power analysis; in most cases the best known prior solutions included manual parameter tuning by the macro owner. The first pass of SynTunSys resulted in a 36% improvement in total negative slack, a 60% improvement in worst latch-to-latch slack (macro internal slack), and a 7% power reduction. The actual values of the metrics, summed across all the macros, underscores that the changes in the absolute numbers were significant, e.g., for pass 1, ~780,000 picoseconds of total negative slack was saved across ~200 macros.

Table 2. Average SynTunSys improvement over best known prior solution based on post-route timing and power analysis.

SynTunSys Pass (Chip Release)	Latch-to-Latch Slack	Total Negative Slack	Total Power
Improvement %	(%)	(%)	(%)
pass 1	60%	36%	7%
pass 2	24%	2%	3%
Sum of 200 macros	(ps)	(ps)	(arb. units)
pre-pass 1	-1929	-2150385	17770
post-pass 1	-765	-1370731	16508
Sum of 25 macros	(ps)	(ps)	(arb. units)
pre-pass 2	-260	-185087	3472
post-pass 2	-198	-180896	3379

A second SynTunSys tuning run by the macro owners to further improve timing and power was performed during the second chip release. These second-pass tuning runs build off the prior tuning run results to further search the design space. In some cases the macro logic was also quite different in the second chip release, leading to a different design space. For the second tapeout we had a less controlled study, but wider usage by macro owners. Based on data from 25 macros (Table 2, "pass 2") we still see considerable improvements on macros after the second pass of SynTunSys.

V. Conclusion

To our knowledge, SynTunSys is the first self-evolving and autonomous system for tuning the input parameters of logic and physical synthesis tools. By taking over the process of tuning the input parameters and by learning automatically from the information of previous synthesis runs, SynTunSys realizes a new level of abstraction between designers and tool developers. The application of SynTunSys to the IBM z13 22nm high-performance server processor yielded on average a 36% improvement in total negative slack and a 7% power reduction. This work opens new avenues of research such as automatic feature extraction of parameters and primitives for design-space reduction and enhanced mining of archived data to improve scenario combinations.

Acknowledgments

This work is partially supported by the NSF (A#: 1527821) and by C-FAR (C#: 2013-MA-2384), one of the six SRC STARnet centers.

References

[1] J. D. Warnock, et al., "22nm Next-Generation IBM System z Microprocessor," ISSCC 2015.

[2] O. Azizi, et al., "An Integrated Framework for Joint Design Space Exploration of Microarchitecture and Circuits," DATE 2010.

[3] S. Xydis, et al., "A Meta-Model Assisted Coprocessor Synthesis Framework for Compiler/Architecture Parameters Customization," DATE 2013.

[4] H.-Y. Liu and L. P. Carloni, "On Learning-Based Methods for Design-Space Exploration with High-Level Synthesis," DAC 2013.

[5] M. K. Papamichael, P. Milder, J. C. Hoe, "Nautilus: Fast Automated IP Design Space Search Using Guided Genetic Algorithms," DAC 2015.

[6] M. M. Ziegler, et al., "POWER8 Design Methodology Innovations for Improving Productivity and Reducing Power," CICC 2014.

[7] L. Trevillyan, et al., "An Integrated Environment for Technology Closure of Deep-Submicron IC Designs," IEEE Design & Test of Computers, vol. 21:1, pp. 14-22, 2004.

[8] G. Fursin, et al., "Milepost GCC: Machine Learning Enabled Self-tuning Compiler," International Journal Parallel Programming, 39:296-327, 2011.

[9] G. Beltrame et al., "Decision-Theoretic Design Space Exploration of Multiprocessor Platforms," IEEE TCAD, 29(7):1083–1095, July 2010.

[10] Z. Wang et al., "Bayesian Optimization in High Dimensions via Random Embeddings," Int'l Joint Conf. on Artificial Intelligence (IJCAI-13), 2013.

[11] M. M. Ziegler, G. D. Gristede, V. V. Zyuban, "Power Reduction by Aggressive Synthesis Design Space Exploration," ISLPED 2013.