

Automatic DRAM Subsystem Configuration with irace

Lukas Steiner
Microelectronic Systems Design
Research Group,
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
lukas.steiner@rptu.de

Gustavo Delazeri
Microelectronic Systems Design
Research Group,
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
delazeri@rptu.de

Iron Prando da Silva
Fraunhofer Institute for Experimental
Software Engineering (IESE)
Kaiserslautern, Germany
iron.prandodasilva@iese.fraunhofer.de

Matthias Jung
HTW Saar
Saarbrücken, Germany
Fraunhofer Institute for Experimental
Software Engineering (IESE)
Kaiserslautern, Germany
matthias.jung@iese.fraunhofer.de

Norbert Wehn
Microelectronic Systems Design
Research Group,
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
norbert.wehn@rptu.de

ABSTRACT

Nowadays, DRAM subsystem configuration includes a large number of parameters, resulting in an extensive design space. Setting these parameters is a challenging step in system design as the parameter-workload interactions are complex. Since design space exploration by exhaustive simulation is infeasible due to limited computing resources and development time, semi-automatic configuration involving both manual as well as simulation-based decisions is state-of-the-art. However, it requires a lot of expertise in the DRAM domain as well as application knowledge, and there is no guarantee for a good performance of the resulting subsystem. In this paper, we present a new framework that fully automatizes the DRAM subsystem configuration for a given parameter space and set of target applications. It is based on irace, a software package originally developed for automatic configuration of optimization algorithms. We show that the framework finds nearly-optimal configurations, while only a fraction of all application-configuration combinations has to be evaluated. In addition, all returned configurations perform better than a predefined standard configuration. Thus, our framework enables designers to automatically determine a suitable DRAM subsystem for their platform.

CCS CONCEPTS

• **Hardware** → **Software tools for EDA**; **Dynamic memory**.

KEYWORDS

DRAM, Design Space Exploration, irace, Automatic Algorithm Configuration

1 INTRODUCTION

The recent increase in memory-intensive applications (e.g., machine learning) has brought the DRAM subsystem more into the design focus [12]. To fulfill the high bandwidth and low latency requirements of these applications, dedicated memory subsystems either have to be newly designed or, more commonly, assembled from off-the-shelf DRAM devices and memory controller IP. Both approaches are challenging as they involve the specification of a large number of parameters to create a configuration that meets all requirements. As shown in Figure 1, this includes different channel organizations like the number of ranks, device characteristics like the DRAM standard or storage density, and controller characteristics like the scheduling policy. Therefore, it is important to efficiently explore the available design space in order to find an appropriate subsystem configuration for a given set of applications. Virtual prototypes have proven to be a useful aid in this context, since they allow to compare different parameter settings in a short time and without the need for real hardware, which is particularly helpful in early design phases [25]. However, the number of possible configurations grows exponentially with the number of parameters, so the point at which exhaustive simulation becomes infeasible for determining an optimal configuration is quickly reached. As an example, with only two settings for each of the parameters shown in Figure 1, the number of possible configurations is already $2^{15} = 32768$. Combined with different example workloads, the required simulations for an exhaustive search quickly ends up in the hundred thousands or millions. For this reason the configuration process usually relies at least partially on manual decisions taken by designers in order to restrict the considered design space. The problem with manual decisions is that nowadays the combined DRAM-workload behavior is difficult to intuitively determine [9], so a lot of expertise in the DRAM domain as well as application knowledge is required.

Especially when the DRAM subsystem is only purchased as one component of a larger platform, designers often lack this expertise. A fully automatic parameter configuration process would prove helpful in these cases.

In the field of optimization there exists a related problem called *Algorithm Configuration Problem* (ACP) [8]: given a parameterized algorithm, find the setting of its input parameters to optimize its performance for a set of possible problem instances to be solved while meeting several constraints [3, 16]. Since manual parameter tuning has the already mentioned disadvantages, it is increasingly replaced by so-called *automatic algorithm configuration*. This approach is divided into two phases. In a primary tuning phase a configuration is automatically chosen based on a training set representative for the particular problem to be solved by the parameterized algorithm, and in a secondary production phase the chosen configuration is applied to unseen instances of the same problem. Manual influence is completely eliminated in the search process. The open-source software package *irace* [16] implements *iterated racing* procedures, which have proved particularly successful in automatic algorithm configuration [15, 20, 23]. The main advantage of *irace* over an exhaustive search is that the minimum number of required configuration evaluations only depends logarithmically on the number of parameters, which allows to handle large parameter spaces as well. In this paper, we show that the DRAM subsystem configuration problem can be mapped to an instance of the ACP. This enables us to use *irace* for a completely automatic DRAM subsystem configuration, given a parameter space and a set of target applications. To evaluate different parameter configurations on training instances, we couple *irace* with the open-source DRAM simulator DRAMSys [26]. This simulator supports a very large DRAM configuration space, offers a high modelling accuracy, and achieves high simulation speeds. We demonstrate the efficiency of our new approach by an exemplary application.

In summary, the paper makes the following new contributions:

- We present a framework for automatic DRAM subsystem configuration that couples the *irace* package with DRAMSys.
- We use the framework to search an appropriate memory controller configuration for a given set of application instances.
- We show that already a small subset of training instances representative for the target applications is sufficient to find nearly-optimal configurations.
- We show that even with the smallest possible evaluation budget all configurations found by our framework perform better than a typical standard configuration.

The remaining paper is structured as follows. Section 2 discusses related work. Section 3 introduces relevant background information. In Section 4 the developed framework is presented. After an explanation of the experimental setup in Section 5 the corresponding results are shown in Section 6. Section 7 concludes the paper and provides an outlook on future work.

2 RELATED WORK

DRAM design space exploration is a very challenging topic and an active research field. The authors of [27] have reviewed 3D-stacked DRAMs with respect to performance, energy and area efficiency to find the optimal device architecture for various DRAM technology

nodes. [7] explores 3D-stacked memories, but the focus is on different in-memory accelerator implementations and their trade-offs with respect to area and power; similar research is conducted in [24] with respect to frequency and cache sizes of the accelerators. [9] presents an experimental study of the interaction between different DRAM types and workloads. From the experiments, several key observations are drawn, which can aid designers to choose the right DRAM standard for their platform. [21] focuses on the memory controller's address mapping and proposes a methodology to find an optimal mapping for a single fixed memory access pattern, which maximizes energy efficiency as well as sustainable bandwidth. Since the number of possible mappings is too large to try out all of them, the problem is solved efficiently with a branch and bound algorithm. [10] introduces a selection and configuration methodology for DRAM system-level parameters in real-time mobile systems. In the work, three different low-power DRAM standards are analyzed considering worst-case bandwidth requirements and power consumption requirements. The methodology defines distinct steps, which are executed manually to find the final configuration. The authors of [2] have examined hybrid main memory solutions for high-capacity and low-power Pareto optimization targets. All of the presented works only consider a small subset of the DRAM design space (e.g., DRAM standard or address mapping) and, thus, allow for exhaustive simulations or manual analysis of results. In [1] automatic design generation techniques are used to find Pareto-optimal hardware implementations of DRAM-optimized FFT algorithms. However, in this case the DRAM parameters are fixed and only the remaining hardware is generated automatically. There exist some automatic design space exploration approaches for processors and complete systems. [14] proposes a methodology to automatically tune processor design parameters to application-specific requirements. SystemCoDesigner [11] automatically generates hardware accelerators from behavioral SystemC models and performs an automatic design space exploration, in which the accelerators are assembled to an overall system. Both [18] and [22] focus on system-level design space exploration, but perform this step on a much higher abstraction level.

3 BACKGROUND

In this section, we give an overview of the DRAM design space and present an informal discussion of the ACP, iterated racing and *irace*. For a more formal and complete treatment, please refer to [4, 16].

3.1 DRAM Design Space

As indicated in the introduction, the DRAM design space comprises a very large number of configuration parameters. It can be split up into the organization part, which specifies the composition of the DRAM subsystem from channels, devices and ranks, the device characteristics like DRAM standard or storage density, and the controller configuration, e.g., scheduling policy or refresh mode. An overview of the most common DRAM subsystem parameters is provided in Figure 1. Each one of these parameters influences the behavior of the DRAM subsystem and, thus, the performance of an application that is accessing the memory. The goal of design space exploration is to tailor the DRAM subsystem to specific platform

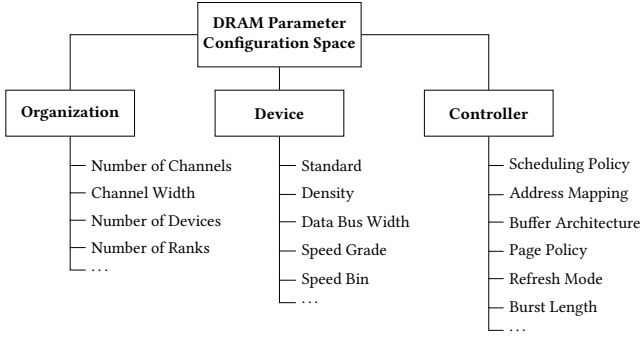


Figure 1: Overview of DRAM Parameter Configuration Space

and application requirements and find a nearly-optimal parameter configuration with respect to a target metric, e.g., application runtime, bandwidth utilization or power consumption. The design space exploration can also be divided into multiple consecutive phases, where each phase only considers a subset of the available parameters. Early phases focus more on the organization part, while in later phases the selected hardware is configured appropriately. DRAMSys [26] is an open-source DRAM subsystem simulator that supports a very large number of different configuration parameters from the space shown in Figure 1 and achieves both a high result accuracy as well as high simulation speeds, making it ideal for design space exploration.

3.2 The Algorithm Configuration Problem

Let A be a configurable algorithm with parameters $p_1, p_2, \dots, p_n$. Each parameter p_i can take different values from a given domain. An assignment of values to each parameter p_i is called a *configuration*, and is denoted by θ . The set of all possible configurations of A is denoted by Θ and is called the *parameter* or *configuration space* of A . The purpose of an algorithm is to solve a problem, and a problem is usually represented by an instance. Let I be a finite set of instances for algorithm A , and $c : I \times \Theta \rightarrow \mathbf{R}$ be a performance function that, given an instance $i \in I$ and a configuration $\theta \in \Theta$, returns the performance of algorithm A parameterized by θ on instance i . We call an *optimization scenario* (or just *scenario*, when it is clear by the context) the tuple (A, Θ, c, I) , where A is our algorithm, Θ is its configuration space, c is the performance function and I is a set of instances.

Given an optimization scenario $S = (A, \Theta, c, I)$, the Algorithm Configuration Problem (ACP) can be stated as: find a configuration $\theta^* \in \Theta$ such that

$$\theta^* = \arg \min_{\theta \in \Theta} F(\theta) \quad (1)$$

where F is a function that measures the aggregated performance of any configuration $\theta \in \Theta$ on the set of instances I . As an example, F could measure the worst performance obtained by a configuration, i.e., $F(\theta) = \max \{c(i, \theta), i \in I\}$, or the average performance, that is, $F(\theta) = (1/|I|) \sum_{i \in I} c(i, \theta)$.

3.3 Iterated Racing and irace

At first glance, it might seem that the ACP could be solved by running all configurations on all instances and then selecting the

optimal one. In practice, however, this is usually infeasible due to the size of the parameter space and the time taken to run the algorithm. This motivates the use of search procedures that explore only some parts of the parameter space. Although these procedures are usually not guaranteed to find the optimal configuration, they tend to yield configurations that are much closer to the optimum than a randomly selected one. In this context, iterated racing [3, 16] is a method to tackle the ACP. Iterated racing can be viewed as a three-step process: (1) sampling new configurations from a sampling distribution, (2) selecting a set of configurations by racing [5], and (3) biasing the sampling distribution towards the best configurations found in step (2). The precise form of the sampling process, the racing procedure, and the biasing of the sampling distribution are not specified and depend on the implementation. The irace package is an implementation of iterated racing and will be used in this work. In the next paragraphs, we give more details about how racing and the sampling process are implemented in irace. For a complete description of the functionality, please refer to [16].

Let us start by describing the general racing process. Suppose we have a set $C = \{\theta_1, \theta_2, \dots, \theta_n\}$ of candidate configurations and let I be the set of instances. At each step of the race, we execute all candidate configurations C , possibly in parallel, over a single instance $i \in I$. After a predetermined number of iterations, we apply a statistical test (e.g., Friedman test [6]) to check if any configuration in the race performed statistically worse than any other configuration on the instances seen so far. If that is the case, then the configuration is eliminated from the race. The race continues until the number of surviving configurations is less than or equal to a predetermined constant, the instance set is exhausted, or the maximum number of algorithm evaluations is exceeded.

After each race, we have a set of surviving configurations. We call these configurations elite configurations and denote them by Θ^{elite} . To generate a new configuration, an elite configuration is first selected by sampling a discrete probability distribution that assigns higher probabilities to better-ranked elites (according to the previous race). In irace, each parameter of a configuration has an associated probability distribution, hence the sampling of a new configuration occurs in a parameter-wise manner. If the parameter is numerical, the associated distribution is a normal distribution centered at the parameter value and with a standard deviation that decreases at each iteration to increase the bias towards similar configurations. If the parameter is categorical, its associated distribution is a discrete distribution that is also updated at each iteration to increase the bias.

The procedures described in the last two paragraphs form the core of the irace implementation of iterated racing and are repeated until a predetermined number of calls of the target algorithm is reached. This number is referred to as *budget* and is denoted by B .

4 AUTOMATIC DRAM SUBSYSTEM CONFIGURATION

As explained in Section 3.1, the goal of DRAM design space exploration is to find a subsystem configuration that optimizes a target metric for a specific set of applications. From the definitions given in Section 3.2, we see that this task can be mapped to an instance

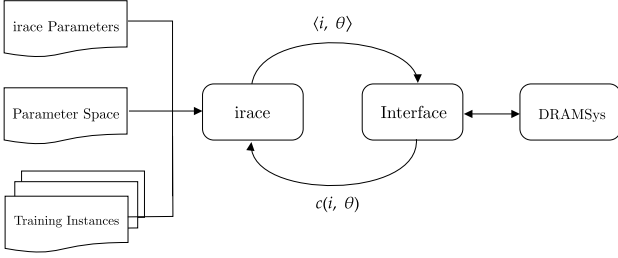


Figure 2: Overview of the Optimization Process

of the ACP. DRAMSys can be viewed as a parameterizable algorithm, where configurations represent different design choices (e.g., number of channels, scheduling algorithm, etc.). Instances are represented by target applications that should be executed on the target platform. They can be provided to DRAMSys by coupling the simulator with a core model and running the applications on it or in the form of prerecorded memory access traces. Given a configuration and an application, one of the many statistics provided by DRAMSys at the end of a simulation can be used as a performance function (e.g., energy consumption, average response latency, simulation time, etc.). Lastly, the aggregated performance function F depends on what we are optimizing for. As an example, if $c(i, \theta)$ returns the energy consumption and the goal is to minimize the average energy consumption, $F(\theta)$ could be equal to $(1/|I|) \sum_{i \in I} c(i, \theta)$.

To apply this approach, we built a framework that couples irace with DRAMSys. The framework allows the specification of a parameter space, a set of training instances and a tool setup, which are then provided to irace. Every time irace needs to know the performance of a configuration θ on an instance i , a call is made to an interface that is responsible for running DRAMSys parameterized by θ on instance i . This interface also collects the statistics that DRAMSys outputs and then returns the appropriate value, i.e., $c(i, \theta)$, to irace. When the specified budget is exhausted, the best configuration found is returned. An overview of this workflow is shown in Figure 2.

5 EXPERIMENTAL SETUP

To enable a comparison of results returned by irace to the optimal configuration, we set up a scenario with a parameter space small enough to be exhaustively searched. This also allows us in a first step to rank each possible configuration according to the aggregated performance function. In a second step, the scenario is then fed into our optimization framework. To assess the robustness of our method, we execute irace under several distinct conditions (e.g., different budgets and different seeds). Before a DRAM subsystem can be optimized, the input parameters for our framework need to be defined. This includes the instance set I over which the optimization will take place (target applications), the parameter space of our DRAM subsystem, the performance function $c(i, \theta)$ as well as the aggregated performance function $F(\theta)$, and the additional setup parameters for irace. Besides that, we also define a typical standard configuration that will be used as a baseline for our results.

5.1 Instance Set

In this work, we consider I to be a set of 119 prerecorded dense memory access traces representing computations of commonly used image manipulations [13, 21]. We manually selected a subset of instances I_{train} composed of 26 instances ($\approx 22\%$ of $|I|$) that represent the universe of 119 instances well. This subset is given to irace during the optimization process.

The complete set of instances I is as described below:

- 2D rotation: a total of 49 instances corresponding to 90° rotation of 2D images with multiple image resolutions and bit depths
- 3D rotation: 30 instances of 3D image rotation with multiple image resolutions and bit depths
- 2D filter: a total of 40 instances of convolution operations over 2D images with multiple image resolutions, kernel sizes and bit depths

5.2 DRAM Subsystem Configuration Space

We consider the optimization of the memory controller configuration of an HBM2 subsystem. It is inspired by the Xilinx Virtex UltraScale+ HBM2 subsystem [19], but reduced to a single channel to achieve a higher bandwidth utilization. The considered parameters and their possible options are listed below:

- Page policy: open, open adaptive, closed, closed adaptive
- Refresh policy: all bank, per bank
- Address mapping: pseudo channel-row-bank group-bank-column and the same one with an interleaved bank group bit
- Scheduler: FIFO, FIFO within banks, FR-FCFS, FR-FCFS with write watermark

In total, this set of parameters allows for 64 different configurations. Thus, it is small enough to exhaustively search for the optimal one considering the 119 DRAM traces introduced in Section 5.1.

5.3 Performance Functions

For our demonstration scenario, the selected performance function is the total simulation time (not to be confused with the wall-clock time), i.e., given an instance i and a configuration θ , $c(i, \theta)$ will return the time taken to process all memory requests in i when the DRAM subsystem is parameterized by θ . Since our main goal is to find a configuration that minimizes the total required time to process a batch of related tasks, a natural choice for the aggregated performance function is the sum of all performances across the instance set, i.e., $F(\theta) = \sum_{i \in I} c(i, \theta)$.

5.4 irace Setup

Concerning irace’s behavior, there are four main elements that need to be considered. The first and most important one is the budget. We define three budget values based on the 1664 algorithm executions that would be required by a hypothetical exhaustive search executed only on the set of training instances I_{train} . This budget is called $B_{\text{exhaustive}}$. B_{min} is the minimum budget required by irace for the parameter space defined in Section 5.2 (in our case $B_{\text{min}} = 180$). We consider three budget values: B_{min} , $B_{\text{min}} + 0.15 \cdot B_{\text{exhaustive}}$, and $B_{\text{min}} + 0.30 \cdot B_{\text{exhaustive}}$.

The second and third elements we have to consider deal with the stochastic nature of irace. Given that the generation of new configurations is done by random sampling, we experiment with three different seed values. Not only that, but the order in which instances are presented to irace is also important, since different instances may influence which configurations are discarded. To account for that, we experiment with three different random orderings of the instances in I_{train} .

Finally, the last element is the statistical test that will be used during the race. The irace package offers more than one option for the statistical test, and the option that best fits our goal is the Student's t-test [17], which uses the mean performance over the seen instances to compare two configurations. Any other hyperparameter not described here was set to the default value. In summary, the complete evaluation of all the combinations of the four elements described above required 27 executions of irace.

5.5 Standard Configuration

Table 1 shows the configuration we chose as our baseline. It is similar to the default configuration described in [19] and optimized for sequential memory accesses, which predominate in the examined applications. In general, our approach is not limited to this application type, but more random memory accesses offer less optimization potential.

Table 1: Standard Configuration

Parameter	Value
Page Policy	Open Adaptive
Refresh Policy	All Bank
Address Mapping	Interleaved
Scheduler	FR-FCFS

6 RESULTS

All experiments were performed on an Intel Xeon Silver 4210R CPU (10C/20T) and 32 GB of DDR4 memory. The exhaustive search evaluates a total of 7616 combinations of instance and configuration, i.e., 64 possible configurations evaluated over the 119 instances. Evaluations were distributed across all 20 threads, with a total search wall-clock time of 10 hours and 38 minutes, which is in average 5.0 seconds per single evaluation. We also allowed irace to use all 20 threads in parallel.

Table 2: Average irace Wall-Clock Execution Times

Budget	Mean	Std. Dev.	Mean ESWT	Mean Budget
180	50 min	22 min	8 %	16.7 s
430	81 min	30 min	13 %	11.3 s
680	100 min	38 min	16 %	8.8 s

Table 2 shows the average execution times and the corresponding standard deviations of the irace executions per budget, as well as their proportions to the exhaustive search wall-clock time (ESWT) and the average execution time per single evaluation. We see that, as one would expect, irace uses only a fraction of the ESWT. However,

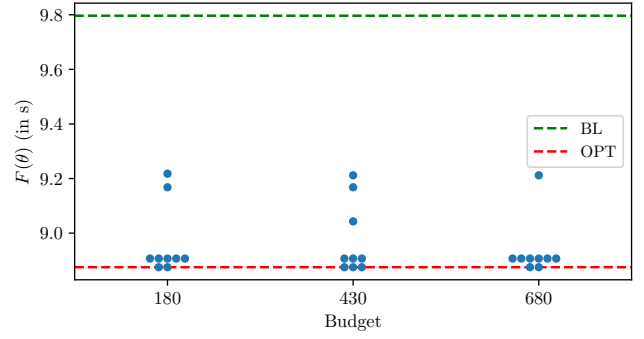


Figure 3: irace Best Found Configurations Statistics

Table 3: Average Value of $F(\theta)$ for each irace Trial

Budget	Mean	Std. Dev.	Mean - OPT OPT	Mean - BL BL
180	8.96 s	0.13 s	1 %	-9 %
430	8.97 s	0.13 s	1 %	-8 %
680	8.93 s	0.11 s	1 %	-9 %

when the execution times are set in relation to the number of evaluations, irace requires much more time for a single evaluation than the exhaustive search, especially for smaller budgets. The reason for this behavior is that the degree of parallelism achieved by irace is limited by the number of sampled configurations and the application of the statistical test. Hence, unlike the exhaustive search, irace cannot make use of all 20 threads during the entire search. It should be noted that with a larger configuration space irace would outperform an exhaustive search more significantly in terms of execution time because it could achieve a better CPU utilization, and the minimum required budget $B_{\min}$ as well as the minimum number of iterations only depend logarithmically on the number of parameters.

As stated in Section 5, our goal is to find a DRAM subsystem configuration that minimizes the total simulation time taken to process a sequence of image manipulation tasks. Using the information gathered by the exhaustive search, we computed $F(\theta)$ for each possible configuration θ (remember that in Section 5.3 we defined $F(\theta)$ to be equal to $\sum_{i \in I} c(i, \theta)$, the sum of the simulation times obtained across all 119 instances). We can use this information to find out which of the 64 possible configurations is the optimal one for our metric and how the configurations found by irace compare with the universe of all possible configurations.

Figure 3 shows the distribution of the value obtained for $F(\theta)$ by the best configuration found in each irace trial. The green line marks the value of $F(\theta)$ obtained by the baseline configuration (BL) introduced in the last section and the red line marks the best value (OPT) of $F(\theta)$ obtained by any of the 64 possible configurations. From the figure, we see that all the configurations found by irace perform better than our baseline, even when the budget is set to the minimum required value. It is also worth mentioning that in 7 of the 29 irace trials the optimal configuration was found.

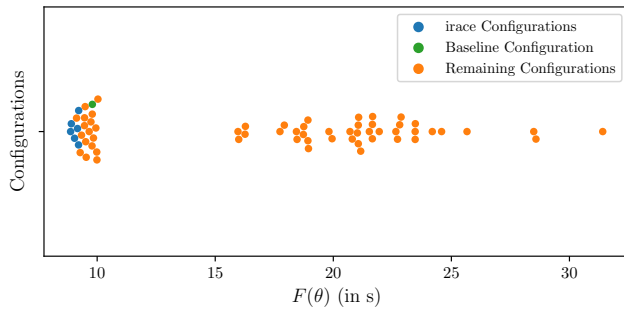


Figure 4: Configuration Space Evaluation

Table 3 shows the average value of $F(\theta)$ obtained by each irace trial grouped by budget, the corresponding standard deviations and the deviations from the optimal configuration as well as the baseline. In our example scenario, the improvement for higher budget values is only marginal (the medium budget performs even worse than the smaller one), and in all three cases, we are on average just 1 % away from the optimality. In principle, however, higher budget values decrease the variance across all runs with different combinations of seed value and instance ordering, achieving good results more consistently. By looking at the fifth column, we can see that, on average, we achieve an 8 % to 9 % improvement over the baseline.

To assess how the space of best configurations found by irace relates to the complete configuration space of all 64 configurations, we present Figure 4. In this plot, each point corresponds to one configuration. The x-axis represents the aggregated performance function $F(\theta)$ corresponding to the sum of total simulation times, while the y-axis is only used to distribute the configurations. Configurations that were the best-found configuration in one or more of the 27 irace trials are painted in blue (several irace trials found the same configuration, hence the number of blue dots is less than 27). The baseline configuration is painted in green. As we can see, the baseline configuration is in the cluster of configurations that are close to the optimal one, which means that it already is a top performer. Nevertheless, as we have already seen in Figure 3, all of the configurations found by irace still outperform the baseline, which shows the robustness of our method.

7 CONCLUSION

In this work we showed that automatic algorithm configuration techniques can be applied in DRAM design space exploration to find an appropriate subsystem configuration for a given set of applications. For this purpose, we developed a framework that couples irace with DRAMSys and used this framework to automatically explore the parameter space of an exemplary HBM2 memory subsystem. Using the framework we were able to automatically find nearly-optimal results while only a small number of all possible application-configuration combinations had to be evaluated. In the future, we will extend our framework to the compute side to automatically configure complete systems.

ACKNOWLEDGMENTS

This work was partly funded by the German Federal Ministry of Education and Research (BMBF) under grant 16ME0717 (MANNHEIM-MEMTONOMY) and supported by the Fraunhofer High Performance Center for Simulation- and Software-Based Innovation.

REFERENCES

- [1] Berkin Akin, Franz Franchetti, and James C. Hoe. 2014. Understanding the design space of DRAM-optimized hardware FFT accelerators. In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. 248–255. <https://doi.org/10.1109/ASAP.2014.6868669>
- [2] Saeideh Alinezhad. 2022. Exploring Pareto-optimal Hybrid Main Memory Configurations using different Emerging Memories. (7 2022). <https://doi.org/10.36227/techrxiv.20326149.v1>
- [3] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. 2007. Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. In *Hybrid Metaheuristics*, Thomas Bartz-Beielstein, Maria José Blesa Aguilera, Christian Blum, Boris Naujoks, Andrea Roli, Günter Rudolph, and Michael Sampels (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 108–122.
- [4] Mauro Birattari. 2009. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer Berlin, Heidelberg, Germany.
- [5] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. 2002. A Racing Algorithm for Configuring Metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (New York City, New York) (GECCO'02). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 11–18.
- [6] W.J. Conover. 1999. *Practical nonparametric statistics, third edition*. John Wiley & Sons, New York, NY.
- [7] João Paulo C. de Lima, Paulo Cesar Santos, Marco A. Z. Alves, Antonio C. S. Beck, and Luigi Carro. 2018. Design Space Exploration for PIM Architectures in 3D-Stacked Memories. In *Proceedings of the 15th ACM International Conference on Computing Frontiers* (Ischia, Italy) (CF '18). Association for Computing Machinery, New York, NY, USA, 113–120. <https://doi.org/10.1145/3203217.3203280>
- [8] Katharina Eggensperger, Marius Lindauer, and Frank Hutter. 2019. Pitfalls and Best Practices in Algorithm Configuration. *J. Artif. Int. Res.* 64, 1 (jan 2019), 861–893. <https://doi.org/10.1613/jair.1.11420>
- [9] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. 93–93.
- [10] Manil Dev Gomony, Christian Weis, Benny Akesson, Norbert Wehn, and Kees Goossens. 2012. DRAM selection and configuration for real-time mobile systems. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 51–56. <https://doi.org/10.1109/DATE.2012.6176432>
- [11] Christian Haubelt, Thomas Schlichter, Joachim Keinert, and Mike Meredith. 2008. SystemCoDesigner: Automatic design space exploration and rapid prototyping from behavioral models. In *2008 45th ACM/IEEE Design Automation Conference*. 580–585. <https://doi.org/10.1145/1391469.1391616>
- [12] Norman P. Jouppi et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) (ISCA '17). Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3079856.3080246>
- [13] Matthias Jung, Deepak M. Mathew, Christian Weis, Norbert Wehn, Irene Heinrich, Marco V. Natale, and Sven O. Krumke. 2016. ConGen: An Application Specific DRAM Memory Controller Generator. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria, VA, USA) (MEMSYS '16). Association for Computing Machinery, New York, NY, USA, 257–267. <https://doi.org/10.1145/2989081.2989131>
- [14] Prasanna Kansakar and Arslan Munir. 2016. A Design Space Exploration Methodology for Parameter Optimization in Multicore Processors. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 613–618. <https://doi.org/10.1109/ISVLSI.2016.92>
- [15] Manuel López-Ibáñez, Christian Blum, Jeffrey W. Ohlmann, and Barrett W. Thomas. 2013. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing* 13, 9 (2013), 3806–3815. <https://doi.org/10.1016/j.asoc.2013.05.009>
- [16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58. <https://doi.org/10.1016/j.orp.2016.09.002>
- [17] M. López-Ibáñez, L. Pérez Cáceres, J. Dubois-Lacoste, T. Stützle, and M. Birattari. 2022. *The irace package: user guide*. Technical Report. IRIDIA, CoDE, Université Libre de Bruxelles.
- [18] Issam Maalej, Guy Gogniat, Jean Luc Philippe, and Mohamed Abid. 2008. System Level Design Space Exploration for Multiprocessor System on Chip. In *2008 IEEE Computer Society Annual Symposium on VLSI*. 93–98. <https://doi.org/10.1109/>

- ISVLSI.2008.34
- [19] Xilinx. 2021. *AXI High Bandwidth Memory Controller v1.0* (v1.0 ed.).
 - [20] Rafael G. Mesquita, Ricardo M.A. Silva, Carlos A.B. Mello, and Péricles B.C. Miranda. 2015. Parameter tuning for document image binarization using a racing algorithm. *Expert Systems with Applications* 42, 5 (2015), 2593–2603. <https://doi.org/10.1016/j.eswa.2014.10.039>
 - [21] Marco V. Natale, Matthias Jung, Kira Kraft, Frederik Lauer, Johannes Feldmann, Chirag Sudarshan, Christian Weis, Sven Krumke, and Norbert Wehn. 2021. Efficient Generation of Application Specific Memory Controllers. In *The International Symposium on Memory Systems* (Washington, DC, USA) (MEMSYS 2020). Association for Computing Machinery, New York, NY, USA, 233–247. <https://doi.org/10.1145/3422575.3422796>
 - [22] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. 2018. Exact multi-objective design space exploration using ASPmT. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 257–260. <https://doi.org/10.23919/DATE.2018.8342014>
 - [23] Marcella Samà, Paola Pellegrini, Andrea D’Ariano, Joaquin Rodriguez, and Dario Pacciarelli. 2016. Ant colony optimization for the real-time train routing selection problem. *Transportation Research Part B: Methodological* 85 (2016), 89–108. <https://doi.org/10.1016/j.trb.2016.01.005>
 - [24] Marko Scrbak, Mahzabeen Islam, Krishna M. Kavi, Mike Ignatowski, and Nuwan Jayasena. 2015. Processing-in-Memory: Exploring the Design Space. In *Architecture of Computing Systems – ARCS 2015*, Luis Miguel Pinho Pinho, Wolfgang Karl, Albert Cohen, and Uwe Brinkschulte (Eds.). Springer International Publishing, Cham, 43–54.
 - [25] Efsthathios Sotiriou-Xanthopoulos, Sotirios Xydis, Kostas Siozios, George Economakos, and Dimitrios Soudris. 2015. Rapid prototyping and Design Space Exploration methodologies for many-accelerator systems. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 1–2. <https://doi.org/10.1109/FPL.2015.7293990>
 - [26] Lukas Steiner, Matthias Jung, Felipe S. Prado, Kirill Bykov, and Norbert Wehn. 2020. DRAMSys4.0: A Fast and Cycle-Accurate SystemC/TLM-Based DRAM Simulator. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Alex Orailoglu, Matthias Jung, and Marc Reichenbach (Eds.). Springer International Publishing, Cham, 110–126.
 - [27] Christian Weis, Norbert Wehn, Loi Igor, and Luca Benini. 2011. Design space exploration for 3D-stacked DRAMs. In *2011 Design, Automation & Test in Europe*, 1–6. <https://doi.org/10.1109/DATE.2011.5763068>