ASK: Adaptive Sampling Kit for Performance Characterization

Pablo de Oliveira Castro[1],Eric Petit[2],Jean Christophe Beyler[3],William Jalby[1]

[1] Exascale Computing Research,
University of Versailles - UVSQ, France
{pablo.oliveira, william.jalby}@exascale-computing.eu
[2] LRC ITACA, University of Versailles - UVSQ, France
eric.petit@uvsq.fr
[3] Intel Corporation jean.christophe.beyler@intel.com

Abstract. Characterizing performance is essential to optimize programs and architectures. The open source Adaptive Sampling Kit (ASK) measures the performance trade-offs in large design spaces. Exhaustively sampling all points is computationally intractable. Therefore, ASK concentrates exploration in the most irregular regions of the design space through multiple adaptive sampling methods. The paper presents the ASK architecture and a set of adaptive sampling strategies, including a new approach: Hierarchical Variance Sampling. ASK's usage is demonstrated on two performance characterization problems: memory stride accesses and stencil codes. ASK builds precise models of performance with a small number of measures. It considerably reduces the cost of performance exploration. For instance, the stencil code design space, which has more than 31.10^8 points, is accurately predicted using only 1 500 points.

1 Introduction

Understanding architecture behavior is crucial to fine tune applications and develop more efficient hardware. An accurate performance model captures all interactions among the system's elements such as: multiple cores with an out-of-order dispatch or complex memory hierarchies. Building analytical models is increasingly difficult with the complexity growth of current architectures.

An alternative approach considers the architecture as a black box and empirically measures its performance response. The Adaptive Sampling Kit (ASK) gathers many state of the art sampling methods in a common framework simplifying the process. From the samples, engineers build a surrogate performance model to study, predict, and improve architecture and application performance on the design space. The downside of the approach is the exploration time needed to sample the design space. As the number of factors considered grows – cache levels, problem size, number of threads, thread mappings, and access patterns – the size of the design space explodes and exhaustively sampling each combination of factors becomes unfeasible.

To mitigate the problem, the engineer must sample only a limited number of combinations. Moreover, they should be chosen with care: clustering the sampled points in a small portion of the design space biases the performance model. The two fundamental elements of a sampling pipeline are the sampling method and surrogate model.

1. The sampling method decides what combinations of the design space should be explored.
2. The surrogate model extrapolates from the sampled combinations a prediction on the full design space.

Choosing an adequate sampling strategy is not simple: for best results one must carefully consider the interaction between the sampling method and surrogate model [1]. Many implementations of sampling methods are available, but they all use different configurations and interfaces. Therefore, building and refining sampling strategies is difficult. ASK addresses this problem by gathering many state of the art sampling strategies in a common framework. Designed around a modular architecture, ASK facilitates building complex sampling pipelines. ASK also provides reporting and model validation modules to assess the quality of the sampling and find the best experimental setup for performance characterization. The paper's main contributions are:

– ASK, a common toolbox gathering state of the art sampling strategies and simple to integrate with existing measurement tools
– A new sampling strategy, Hierarchical Variance Sampling (HVS), which mitigates sampling bias by using confidence bounds
– An evaluation of the framework, and of HVS, on two representative performance characterization experiments

Section 2 discusses related works. Section 3 explains the HVS strategy. Section 4 succinctly presents ASK's architecture and usage. Finally, Section 5 evaluates ASK on two performance studies: memory stride accesses and 2D stencils.

2 Related Works

There are two kinds of sampling strategies: space filling designs and adaptive sampling. Space filling designs select a fixed number of samples with sensible statistical properties such as uniformly covering the space or avoiding clusters. For instance, Latin Hyper Cube designs [2] are built by dividing each dimension into equal sized intervals. Points are selected so the projection of the design on any dimension contains exactly one sample per interval. Maximin designs [3] maximize the minimum distance between any pair of samples; therefore spreading the samples over the entire experimental space. Finally, low discrepancy sequences [4] choose samples with low discrepancy: given an arbitrary region of the design space, the number of samples inside the region is close to proportional to its measure. By construction, the sequences uniformly distribute points in space. Space filling designs choose all points in one single draw before starting the experiment.

Adaptive sampling methods, on the contrary, iteratively adjust the sampling grid to the complexity of the design space. By observing already measured samples, they identify the most irregular regions of the design space. Further samples are drawn in priority from the irregular regions, which are harder to explore.

The definition of irregular regions changes depending on the sampling method. Variance-reduction methods prioritize exploration of regions with high variance. The rationale is irregular regions are more complex, thereby requiring more measures. Query-by-Committee methods build a committee of models trained with different parameters and compare the committee's predictions on all the candidate samples. Selected samples are the ones where the committee's models disagree the most. Adaptive Multiple Additive Regression Trees (AMART) [5] is a recent Query-by-Committee approach based on Generalized Boosted Models (GBM) [6], it selects non-clustered samples with maximal disagreement. Another recent approach by Gramacy et al. [7] combines the Tree Gaussian Process (TGP) [8] model with adaptive sampling methods [9]. For an extensive review of adaptive sampling methods please refer to Settles [10].

The Surrogate Modeling Toolbox (SUMO) [11] offers a Matlab toolbox building surrogate models for computer experiments. SUMO's execution flow is similar to ASK's: both allow configuring the model and sampling method to fully automate an experiment plan. SUMO focuses mainly on building and controlling surrogate models, offering a large set of models. It contains algorithms for optimizing model parameters, validating the models, and helping users choose a model. On the other hand, most of SUMO's adaptive methods are basic sequential sampling methods. Only a single recent approach is included, which finds trade-offs between uniformly exploring the space and concentrating on nonlinear regions of the space [12]. SUMO is open source but restricted to academic use and depends on the proprietary Matlab toolbox.

ASK specifically targets adaptive sampling for performance characterization, unlike SUMO. It includes recent state of the art approaches that were successfully applied to computer experiments [7] and performance characterization [5]. Simpson et al. [1] show one must consider different trade-offs when choosing a sampling method: affinity with the surrogate model or studied response, accuracy, or cost of predicting new samples. Therefore, ASK comes with a large set of approaches to cover different sampling scenarios including Latin Hyper Cube designs, Maximin designs, Low discrepancy designs, AMART, and TGP. Additionally, ASK includes a new approach, Hierarchical Variance Sampling (HVS).

3 Hierarchical Variance Sampling

Many adaptive learning methods are susceptible to bias because the sampler makes incorrect decisions based on an incomplete view of the design space. For instance, the sampler may ignore a region despite the fact it contains big variations because previous samplings missed the variations.

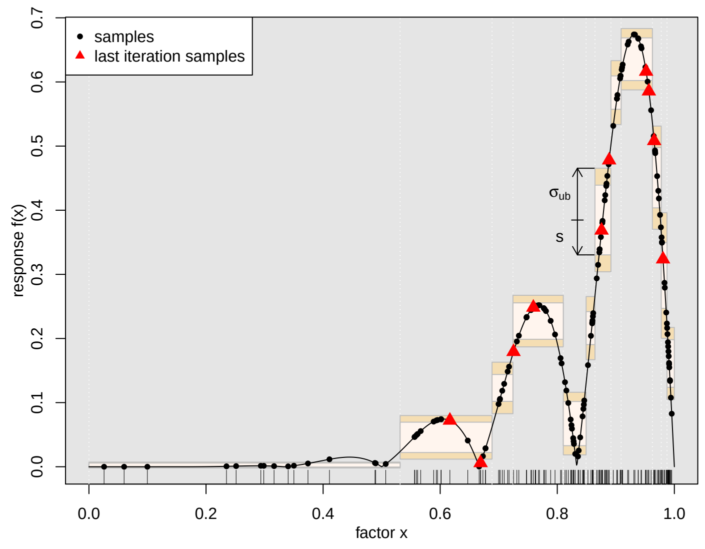


Fig. 1. HVS on a synthetic 1D benchmark after fifteen drawings of ten samples each. The true response, $f(x) = x^5|sin(6.\pi.x)|$, is the solid line. CART partitions the factor dimension into intervals, represented by the boxes horizontal extension. For each interval, the estimated standard deviation, s, is in a light color and the upper bound of the standard deviation, σ_{ub}, is dark. HVS selects more samples in the irregular regions.

To mitigate the problem, ASK includes the new Hierarchical Variance Sampling. HVS' principal concept is to reduce the bias using confidence intervals that correct the variance estimation. HVS partitions the exploration space into regions and measures the variance of each region. A statistical correction depending on the number of samples is applied to obtain an upper bound of the variance. Further samples are then selected proportionally to the upper bound and size of each region. By using a confidence upper bound on the variance, the sampler is less greedy in its exploration but is less likely to overlook interesting regions. In others words, the sampler will not completely ignore a region until the number of sampled points is enough to confidently decide the region has low variance.

HVS is similar to Dasgupta et al. [13] proposing a hierarchical approach for classification tasks using confidence bounds to reduce the sampling bias. Nevertheless, the Dasgupta et al. approach is only applicable to classification tasks with a binary or discrete response; whereas HVS deals with continuous responses, which are more appropriate for performance characterization.

To divide the design space into regions, HVS uses the Classification and Regression Trees (CART) partition algorithm [14] with the Analysis of Variance

(ANOVA) splitting criteria [15] and prunes the tree to optimize cross validation error. At each step, the space is divided into two regions so the sum of the regions variance is smaller than the variance of the whole space. The result of a CART partitioning is shown in Figure 1 where each box depicts a region.

After partitioning, HVS samples the most problematic regions and ignores the ones with low variance. The sampler only knows the empiric variance s^2, which depends on previous sampling decisions; to reduce bias HVS derives an upper bound of the true variance σ^2. Assuming a close to normal region's distribution, HVS computes an upper bound of the true variance σ^2 satisfying $\sigma^2 < \frac{(n-1)s^2}{\chi^2_{1-\alpha/2,n-1}} = \sigma^2_{ub}$ with a $1-\alpha$ confidence[4]. To reduce the bias HVS uses the corrected upper bound accounting for the number of samples drawn.

For each region, Figure 1 plots the estimated standard deviation s, light colored, and upper-bound σ_{ub}, dark colored. Samples are selected proportionally to the variance upper bound multiplied by the size of the region, as shown in Figure 1. New samples, marked as triangles, are chosen inside the largest boxes. HVS selects few samples in the $[0, 0.5]$ region, which has a flat profile.

If the goal of the sampling is to reduce the absolute error of the model, then the HVS method is adequate because it concentrates on high-variance regions. On the other hand, if the goal is to reduce the relative, or percentage, error of the model it is better to concentrate on regions with high relative variance, $\frac{s^2}{\overline{x}^2}$. HVSrelative is an alternate version of HVS using relative variance with an appropriate confidence interval [16]. Section 5 evaluates the two sampling strategies, HVS and HVSrelative, in two performance studies.

4 ASK Architecture

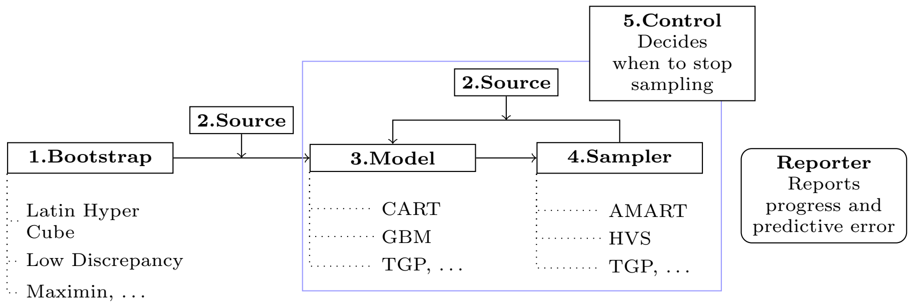


Fig. 2. ASK pipeline

ASK's flexibility and extensibility come from its modular architecture. When running an experiment, ASK follows the pipeline presented in Figure 2:

[4] $1-\alpha = 0.9$ confidence bound is default

1. A *bootstrap* module selects an initial batch of points. ASK provides standard bootstrap modules for the space filling designs described in Section 2: Latin Hyper Cube, Low Discrepancy, Maximin, and Random.
2. A *source* module, usually provided by the user, receives a list of requested points. The source module computes the actual measures for the requested factors and returns the response.
3. A *model* module builds a surrogate model for the experiment on the sampled points. Currently ASK provides CART [14], GBM [6, 17], and TGP [7] models.
4. A *sampler* module iteratively selects a new set of points to measure. Some sampler modules are simple and do not depend on the surrogate model. For instance, the `random` sampler selects a random combination of factors and the `latin` sampler iteratively augments an initial Latin Hyper Cube design. Other sampler modules are more complex and base their decisions on the surrogate model.
5. A *control* module decides when the sampling process ends. ASK includes two basic strategies: stopping when it has sampled a predefined amount of points or stopping when the accuracy improvement stays under a given threshold for a number of iterations.

From the user perspective, setting up an ASK experiment is a three-step process. First, the range and type of each factor is described by writing an experiment configuration file in the JavaScript Object Notation (JSON) format. ASK accepts real, integer, or categorical factors. Then, users write a *source* wrapper around their measuring setup. The interface is straightforward: the wrapper receives a combination of factors to measure and returns their response. Finally, users choose which bootstrap, model, sampler, control, and reporter modules to execute. Module configuration is also done through the configuration file. ASK provides fallback default values if parameters are omitted from the configuration. An excerpt of a two factor configuration with the hierarchical sampler module follows:

```
"factors": [{"name": "image-size",
             "type": "integer",
             "range": {"min": 0, "max": 600}},
            {"name": "stencil-size",
             "type": "categorical",
             "values": ["small", "medium", "large"]}],
"modules": {"sampler": {"executable": "sampler/HVS",
                        "params": {"nsamples":50}}}
```

Editing the configuration file quickly replaces any part of the ASK experiment pipeline with a different module. All the modules have clearly defined interfaces and are organized with strong separation of concerns in mind. It allows the user to quickly integrate custom made modules to the ASK pipeline. For example, by replacing `sampler/HVS` with `sampler/latin` the user replays the same experience with the same parameters but using a Latin Hyper Cube sampler instead.

5 Experimental Study

Two performance characterization experiments were conducted using ASK to achieve two different objectives. The first objective was to validate the ASK pipeline and understand on a low dimension space the behavior of each method using a synthetic microbenchmark called ai_aik. Ai_aik explores the impact of stride accesses to a same array in a single iteration. The design space is composed of 400 000 different combinations of two factors: loop trip N and k-stride. The design space is large and variable enough to challenge sampling strategies. Nonetheless, it is narrow enough to be measured exhaustively providing an exact baseline to rate the effectiveness of the sampling strategies.

The second objective was to validate the strategies on a large experimental space: 2D cross-shaped stencils of varying sizes on a parallel SMP. A wide range of scientific applications use stencils: for instance, Jacobi computation [18, 19] uses 2×2 stencils and high-order finite-difference calculations [20] use 6×6 stencils. The design space is composed of five parameters: the $N \times M$ size of the matrix, the $X \times Y$ size of the stencil, and T the number of concurrent threads used. The design space size has more than 7.10^8 elements in a 8-core system and more than 31.10^8 elements in a 32-core system.

Since an exhaustive measure is computationally unfeasible, the prediction accuracy is evaluated by computing the error of each strategy on a test set of 25 600 points independently measured. The test set contains 12 800 points chosen randomly and 12 800 points distributed in a regular grid configuration over the design space. Measuring the test set takes more than twelve hours of computation on a 32-core Nehalem machine.

All studied sampling strategies use random seeds, which can slightly change the predictive error achieved by different ASK runs. Therefore, the median error, among nine different runs, is reported when comparing strategies.

Experiments ran with six of the sampling strategies included in ASK: AMART, HVS, HVSrelative, Latin Hyper Cube, TGP, and Random. All the benchmarks were compiled with ICC 12.0.0 version. The strategies were called with the following set of default parameters on both experiments:

Samples All the strategies sampled in batches of fifty points per iteration.

Bootstrapping All the strategies were bootstrapped with samples from the same Latin Hyper Cube design, except Random, which was bootstrapped with a batch of random points.

Surrogate Model The TGP strategy used tgpllm [8] model with its default parameters. The other strategies used GBM [6] with the following parameters: `ntrees=3 000, shrinkage=0.01, depth=8`.

AMART ran with a committee size of twenty as recommended by Li et al. [5].

TGP used the Active Learning-Cohn [9] sampling strategy.

HVS, HVSrelative used a confidence bound of $1 - \alpha = 0.9$.

Section 5.1 validates ASK on an exhaustive stride access experiment. Section 5.2 validates ASK on a large design space that cannot be explored exhaustively: multi-core performance of cross-shaped stencils.

5.1 Stride Memory Effects

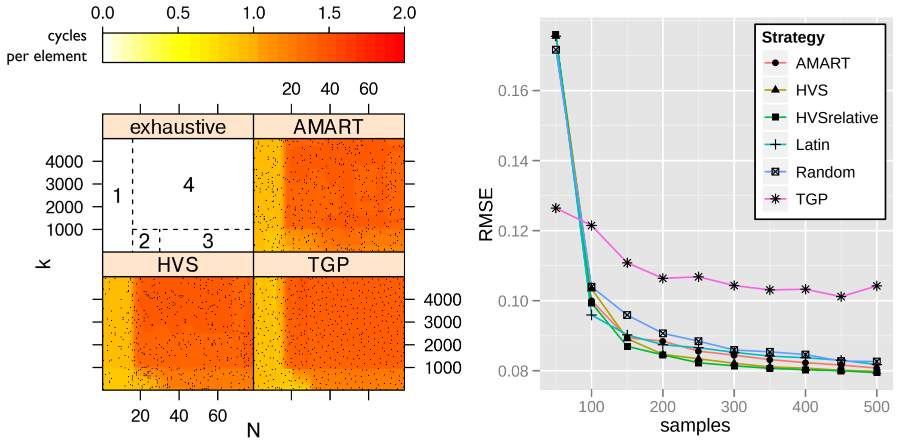


Fig. 3. Stride experiments: (*Left*) exhaustive level plot shows the true response of the studied kernel in cycles per element. AMART, HVS, and TGP respectively show the predicted response of each strategy. Black dots indicate the position of the sampled points. (*Right*) RMSE is plotted for each strategy and number of samples. The median among nine runs of each strategy was taken to remove random seed effects.

This section studies the stride memory accesses of the following ai_aik kernel:

```
for(i=0;i<N*256;i++) {
  res[i]=a[i]+a[i+2*k];
}
```

The baseline is an exhaustive measure of cycle per element performance on a Xeon L5609 quad-core 1.87GHz with 8GB of RAM. The measures revealed the four zones on the left side of Figure 3:

1. In Zone 1 the kernel is fastest because both i and $i + 2.k$ accesses fit in L1.
2. Zone 2 is a transition zone between Zone 1 and Zone 3: under the diagonal, the accessed elements still fit in L1.
3. In Zone 3 accesses do not fit fully in L1 anymore but the performance is still acceptable because the accesses $i + 2.k$ prefetch the data needed for the i accesses in future iterations.
4. In Zone 4, performance is the worst because accesses do not fully fit in L1 and the $2.k$ distance is too wide for efficient software prefetching.

The preceding exhaustive analysis required 400 000 measures of the ai_aik kernel. ASK ran the same experiment with different sampling strategies stopping at five hundred samples. Each method's accuracy was determined by comparing its predictions to the exhaustive baseline.

Comparing the predicted and exhaustive responses in Figure 3 shows that TGP and HVS capture the four zones in detail while AMART is less precise in

Zone 2. The diagonal effect in Zone 2 introduces high variance. Therefore, HVS concentrates its sampling, capturing the zone accurately.

The Root Mean Square Error (RMSE) is the standard metric in the literature to evaluate a model's accuracy. Figure 3, right side, shows the RMSE of each method. The methods, except TGP, are comparable in terms of convergence speed and reached accuracy. TGP scores a poor RMSE performance compared to the other methods. GBM surrogate model seems to be a better fit for this experiment.

In the experiment, ASK's five hundred point sampling successfully captures the performance features of the design space. ASK uses eight hundred times less samples than the exhaustive analysis, while preserving accuracy.

5.2 Stencil Characterization

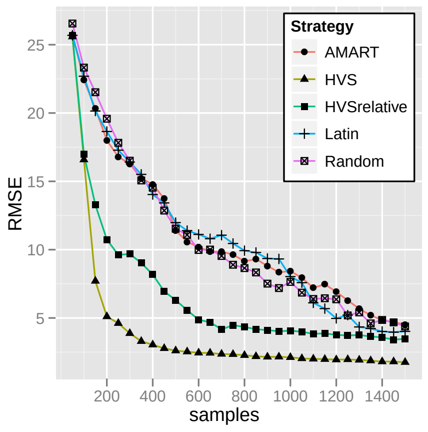
The stencil code characterization is unfeasible with exhaustive exploration, but is possible using ASK's adaptive sampling methods. In the studied stencil code, Figure 4a, five factors are tunable – X and $Y \in \{1, 2, 4, 8, 16\}$ the horizontal and vertical sizes of the stencil, $N \in [64, 2048]$ the number of lines of the matrix, $M \in [64, 2048]$ the number of columns of the matrix, and $T \in [1, 32]$ the number of threads. The stencil was studied on two Nehalem architectures: an 8-core dual-socket Xeon E5620 at 2.40GHz with 24GB of RAM and a 32-core four-socket Xeon X7550 at 2.00GHz with 128GB of RAM. The OpenMP mapping policy was set to `Scatter`. The error was evaluated on an independent test set of 25 600 points.

TGP was not used during the second experiment because it does not handle categorical variables [8]. The computation time needed to select samples with HVS, HVSrelative, and Latin is negligible compared to the time required to measure a batch of samples. AMART is a Query-by-Committee strategy, which generates a prediction on all the candidate points, for twenty different models. In the stencil experiment the number of candidate points is in the order of billions, computing a prediction on all of them is not possible. Therefore, as suggested by Gramacy [8], ASK's AMART implementation reduces the number of candidate points to one thousand with a Latin Hyper Cube presampling.

The sampling strategies' accuracy is measured in terms of RMSE and mean relative error, in Figure 4b. Here only the 32-core results are examined because the sampling strategies' accuracy was similar for both the 8 and 32-core architectures. For RMSE, HVS outperforms all other strategies both in quality of the final model, 1.76 RMSE, and speed of convergence. For mean percentage error, AMART achieves the best final result, 8.89%, followed closely by Latin, Random, and HVSrelative. HVSrelative converges faster than the others.

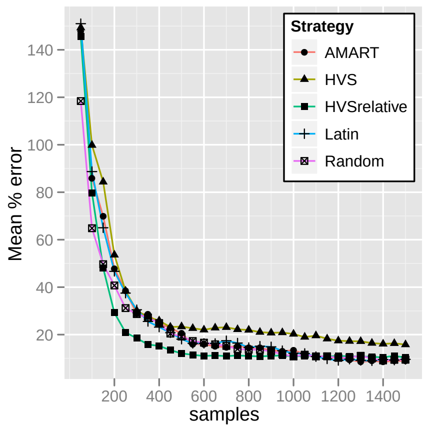
Overall, HVSrelative is the best compromise between RMSE and percentage error because it achieves low final errors and converges quickly to an accurate model. Using only eight hundred samples, HVSrelative predicts the test set with a mean absolute error of 1.51 cycles and a mean relative error of 10.97%.

Figure 4c shows the performance prediction for HVS and HVSrelative on the $X \times 16$ stencils. Each square represents a unique (X, Y, T) configuration. Inside

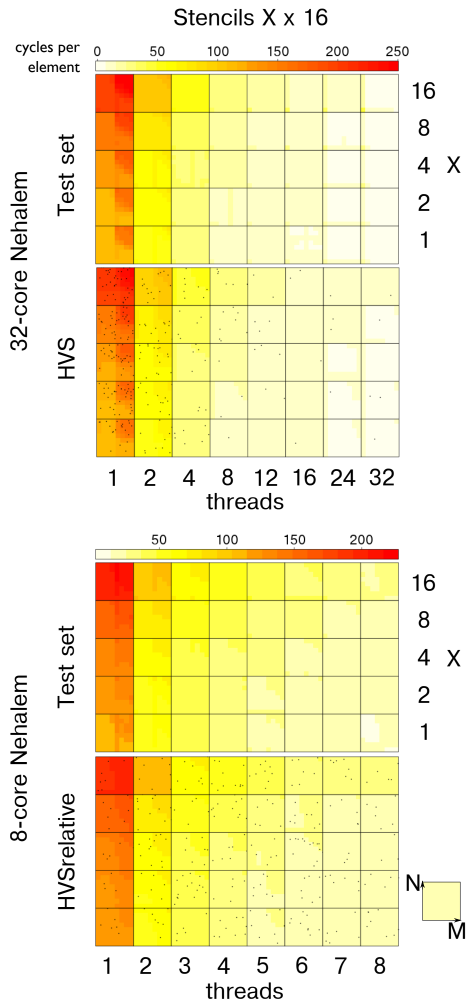


```
#pragma omp parallel for
for(i=Y; i<N-Y; i++)
  for(j=X; j<M-X; j++) {
    for(k=j-X; k<=j+X; k++)
      out[i][j] += in[i][k];
    for(l=i-Y; l<=i+Y; l++)
      out[i][j] += in[l][j];
  }
```

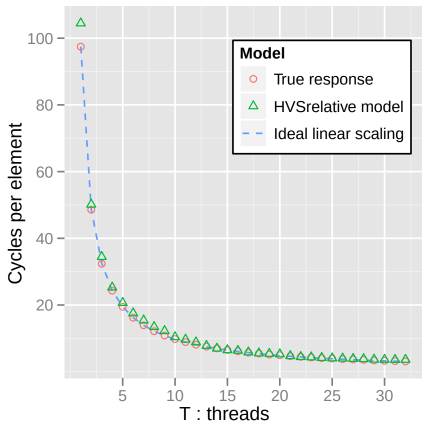
(a) Stencil code evaluated



(b) Error curves for the exploration on 32 cores. The median among nine runs of each strategy was taken to remove random seed effects.



(c) Stencil $X \times 16$ predicted performance vs. test set performance in cycles per element.



(d) Scalability for the 8×8 stencil on a $1\,000 \times 1\,000$ matrix.

Fig. 4. Stencil experiments

each square the performance is plotted depending on the matrix size $N \times M$. The kernel is slowest for high Y stencils whose column order accesses stress the cache. In comparison, X stencil's size impact on performance is negligible. Performance degrades for large matrices, as shown from the darker top-right corners of each square, probably because the matrices exceed the L2 cache capacity. Therefore, choosing an adequate blocking factor should improve the performance. On ASK HVS' 32-core model the mean speed-up obtained by varying the matrix size is 1.65. Using a small matrix with a high number of threads is detrimental because the cost of synchronization predominates.

On both 8 and 32-core targets the stencil code scales linearly up to, respectively, 8 and 32 threads. As an example, the scalability of the application was studied on the 8×8 stencil on a $1\,000 \times 1\,000$ matrix. Figure 4d shows the performance per number of threads predicted by the HVSrelative strategy. The prediction follows the measured true response. The matrix sizes explored fit into the socket's 18Mb L3 cache, additionnaly the 2D stencil beneficiates from data reuse, which explains the strong scaling.

Measuring the whole design space would take centuries whereas ASK adaptive sampling took less than an hour of experiment time with a 1.76 RMSE. ASK is both efficient when dealing with narrow design spaces, as in ai_aik, and large design space, as in the stencil codes experiment.

6 Conclusion

Adaptive sampling techniques drastically reduce exploration time of large design spaces. Nevertheless, choosing the right technique can be difficult for a performance architect. ASK provides an homogeneous interface to multiple state of the art sampling strategies, making the process easier. Adding new strategies to the framework is straightforward due to ASK's modular architecture.

The new HVS strategy reduces experimental bias and is comparable, or even outperforms, other state of the art approaches in two case studies. The stencil code design space, which has more than 31.10^8 points, was accurately predicted using only 1 500 points. The performance characterization field could benefit from adaptive sampling techniques. Hopefully, ASK's open source release will facilitate their adoption.

Currently, users must try different models manually to find what is best suited to their experiment. Automatic model and sampling selection techniques [11, 21] applied to performance experiments will be investigated in future works.

ASK will soon be released at http://code.google.com/p/adaptive-sampling-kit. The experimental data and benchmarks used to produce the paper results are available at the same URL.

Acknowledgments This work has been conducted conjointly by the Exascale Computing Research lab, thanks to the support of CEA, GENCI, Intel, UVSQ, and by the LRC ITACA lab, thanks to the support of the French Ministry for Economy, Industry, and Employment throught the ITEA2 project H4H. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the CEA, GENCI, Intel, or UVSQ.

References

1. Simpson, T., Lin, D., Chen, W.: Sampling strategies for computer experiments: design and analysis. International Journal of Reliability and Applications **2**(3) (2001) 209–240
2. Stein, M.: Large sample properties of simulations using Latin hypercube sampling. Technometrics (1987) 143–151
3. Johnson, M., Moore, L., Ylvisaker, D.: Minimax and maximin distance designs. Journal of statistical planning and inference **26**(2) (1990) 131–148
4. Diwekar, U., Kalagnanam, J.: Efficient sampling technique for optimization under uncertainty. AIChE Journal **43**(2) (1997) 440–447
5. Li, B., Peng, L., Ramadass, B.: Accurate and efficient processor performance prediction via regression tree based modeling. Journal of Systems Architecture **55**(10-12) (2009) 457–467
6. Ridgeway, G.: Generalized Boosted Models: A guide to the gbm package. Update **1** (2007) 1
7. Gramacy, R., Lee, H.: Adaptive design and analysis of supercomputer experiments. Technometrics **51**(2) (2009) 130–145
8. Gramacy, R.: tgp: An R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed gaussian process models. Journal of Statistical Software **19**(9) (2007) 6
9. Cohn, D.: Neural network exploration using optimal experiment design. Neural Networks **9**(6) (1996) 1071–1083
10. Settles, B.: Active Learning Literature Survey. Science **10**(3) (1995) 237–304
11. Gorissen, D., Couckuyt, I., Demeester, P., Dhaene, T., Crombecq, K.: A surrogate modeling and adaptive sampling toolbox for computer based design. The Journal of Machine Learning Research **11** (2010) 2051–2055
12. Crombecq, K., Gorissen, D., Deschrijver, D., Dhaene, T.: A Novel Hybrid Sequential Design Strategy for Global Surrogate Modeling of Computer Experiments. SIAM Journal on Scientific Computing **33** (2011) 1948
13. Dasgupta, S., Hsu, D.: Hierarchical sampling for active learning. In: Proceedings of the 25th international conference on Machine learning, ACM (2008) 208–215
14. Breiman, L., Friedman, J., Olshen, R., Stone, C., Steinberg, D., Colla, P.: CART: Classification and regression trees. Wadsworth: Belmont, CA (1983)
15. Atkinson, E., Therneau, T.: An introduction to recursive partitioning using the RPART routines. Rochester: Mayo Foundation (2000)
16. McKay, A.: Distribution of the Coefficient of Variation and the Extended t Distribution. Journal of the Royal Statistical Society **95**(4) (1932) 695–698
17. Friedman, J.: Greedy function approximation: a gradient boosting machine. Annals of Statistics (2001) 1189–1232
18. Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., Yelick, K.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press (2008) 1–12
19. Treibig, J., Wellein, G., Hager, G.: Efficient multicore-aware parallelization strategies for iterative stencil computations. J. Comput. Science **2**(2) (2011) 130–137
20. Dursun, H., Nomura, K., Peng, L., Seymour, R., Wang, W., Kalia, R., Nakano, A., Vashishta, P.: A multilevel parallelization framework for high-order stencil computations. Euro-Par 2009 Parallel Processing (2009) 642–653
21. Maron, O., Moore, A.: Hoeffding races: Accelerating model selection search for classification and function approximation. Robotics Institute (1993) 263