A Generic C++ Library for Multilevel Quasi-Monte Carlo

R. N. Gantner

Research Report No. 2016-30
June 2016

Funding SNF: 149819
Funding: CSCS Project ID d41

A Generic C++ Library for Multilevel Quasi-Monte Carlo

Robert N. Gantner
Seminar for Applied Mathematics,
ETH Zürich, Rämistrasse 101,
Zürich, Switzerland
robert.gantner@sam.math.ethz.ch

ABSTRACT

We present a high-performance framework for single- and multilevel quasi-Monte Carlo methods applicable to a large class of problems in uncertainty quantification. A typical application of interest is the approximation of expectations of quantities depending on solutions to e.g. partial differential equations with uncertain inputs, often yielding integrals over high-dimensional spaces. The goal of the software presented in this paper is to allow recently developed quasi-Monte Carlo (QMC) methods with high, dimension-independent orders of convergence to be combined with a multilevel approach and applied to large problems in a generic way, easing distributed memory parallel implementation. The well-known multilevel Monte Carlo method is also supported as a particular case, and some standard choices of distributions are included. For so-called *interlaced polynomial lattice rules*, a recently developed QMC method, precomputed generating vectors are required; some such vectors are provided for common cases.

After a theoretical introduction, the implementation is briefly explained and a user guide is given to aid in applying the framework to a concrete problem. We give two examples: a simple model problem designed to illustrate the mathematical concepts as well as the advantages of the multilevel method, including a measured decrease in computational work of multiple orders of magnitude for engineering accuracy, and a larger problem that exploits high-performance computing to show excellent parallel scaling. Some concrete use cases and applications are mentioned, including uncertainty quantification of partial differential equation models, and the approximation of solutions of corresponding Bayesian inverse problems. This software framework easily admits modifications; custom features like schedulers and load balancers can be implemented without hassle, and the code documentation includes relevant details.

PASC '16, June 08–10, 2016, Lausanne, Switzerland
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4126-4/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2929908.2929915

CCS Concepts

•**Mathematics of computing** → **Mathematical software;** *Probabilistic reasoning algorithms;* Dimensionality reduction;

1. INTRODUCTION

Many problems in uncertainty quantification (UQ) for partial differential equations (PDEs) are formulated as expectations over uncertain input parameters. In the case of distributed inputs, these expectations are high-dimensional integrals, which often suffer from the curse of dimensionality. In order to approximate such quantities for large problems, both high-performance computing and state-of-the-art algorithms are essential. This paper describes a generic, high-performance implementation of recently developed methods that can offer immense reductions of both the required computational work and the time to solution.

We consider in this paper a combination of two techniques recently developed in the field of uncertainty quantification for PDEs: multilevel Monte Carlo (MLMC) on the one hand and higher-order quasi-Monte Carlo (HoQMC) quadrature in the form of *interlaced polynomial lattice rules* on the other.

MLMC was developed independently by Giles [17] and Heinrich [23], and aims to improve the convergence rate of the error vs. the work for problems that can be approximated on different levels of accuracy. The goal throughout the recently developed MLMC literature is to achieve a convergence rate of $error \sim work^{-1/2}$, which is optimal when using Monte Carlo (MC) sampling. In certain cases, depending on the convergence rate of a corresponding single-level method, the gains can be significant. However, the rate in terms of the work is limited to $-1/2$ by the use of Monte Carlo. MLMC has been applied to parametric integration [23], path simulation for stochastic differential equations (SDEs) [17], stochastic partial differential equations (SPDEs) [2, 3], and various other problems [1, 15, 35, 36].

The second recent development we consider here is a quasi-Monte Carlo (QMC) method with higher-order convergence rate, by which we mean that the quadrature error decreases like $error \sim N^{-\sigma}$ with $\sigma > 1$, where N is the number of function evaluations. In particular, we focus on so-called interlaced polynomial lattice (IPL) rules, which achieve these higher order rates with $\sigma \geq 2$ for integrands satisfying suitable *sparsity assumptions* [6, 9, 11, 20, 37]. The combination of a multilevel approach with quasi-Monte Carlo has been considered for randomized rank-1 lattice rules in [19] and for the higher-order rules considered here in [10].

Among the advantages of the multilevel approach are its simplicity and generality, both algorithmically and in terms of serial implementation. However, for large problems requiring parallel computing (with possibly "nested parallelism", i.e. when the solver itself is parallelized), things soon become much more complicated. Together with the fact that the computation of approximations to the model in question can require different numbers of CPUs in various cases, such as on different levels or for different realizations of the random field (e.g. when using adaptive mesh refinement), this can greatly complicate a parallel implementation [42, 43].

It is the goal of the library presented here to provide a generic, parallel, high-performance framework which abstracts much of the parallelization details, allowing the user to focus on the application of the multilevel algorithm to his or her problem. The use of C++ and MPI allows this library to be used on the largest parallel computers in order to exploit the excellent parallelization properties of MC and QMC methods. Additionally, routines for the evaluation of IPL rules are provided, allowing their straightforward application to the computational approximation of high-dimensional integrals, also beyond their use in the multilevel methods considered here.

This paper is structured as follows: First, we briefly review the theoretical basics required for applying Monte Carlo (MC) and Quasi-Monte Carlo (QMC) methods to the approximation of statistical moments. In the QMC part, we focus on recently developed higher-order convergent *interlaced polynomial lattice rules*, and mention their advantages over MC and other QMC methods. We then develop the single-level algorithm and its multilevel extension in Sections 2.3 and 2.4, respectively, for both MC and QMC sampling methods, focusing on a formulation amenable to generic implementation.

The generic C++ library, named `gMLQMC`, is described in Section 3, including the parallelization approach and use of MC and QMC rules. A user guide is given in Section 3.4, where the installation and use of the library is briefly explained. A more extensive code documentation is provided with the software itself.

Finally, Section 4 describes two concrete example problems; the first, a simple parametric integral, is designed to elucidate the mathematical properties required for such multilevel methods to be applicable, and how the choice of MC or QMC method influences the resulting convergence rates, illustrating the at times massive gains in accuracy or reductions in computational work that are possible. The second example focuses on the high-performance aspect of the library, and relevant parallel scaling results are shown.

2. THEORY

We consider the problem of computing statistical moments of a *quantity of interest* (abbreviated *QoI* and denoted by ϕ below) depending on an uncertain input. Of interest in applications are often solutions to ordinary or partial differential equation (PDE) models, where the uncertain input can take the form of a coefficient, initial value, or even the domain on which the equation is posed. We assume below that for each realization of the uncertain input, the solution to the model in question can be approximated on a given level, with a certain accuracy.

In order to compute statistical moments, we must approximate expectations, i.e. integrals in the case of continuous state spaces. A typical difficulty is that the uncertain inputs or their parametrizations can be very high-dimensional, for which case Monte Carlo-type methods are well-suited. In Section 2.2, we briefly review Monte Carlo (MC) approximations involving random sampling, as well as quasi-Monte Carlo (QMC) approximations that are based on deterministically chosen samples and can achieve higher rates of convergence.

2.1 Discretization

We refer to the uncertain input as $\boldsymbol{y} \in U$, where the form of $\boldsymbol{y}$ is given by the application at hand. Often, $\boldsymbol{y}$ is a random variable with distribution modeling the behavior of the input. On the other hand, $\boldsymbol{y}$ could be a vector of parameters of an expansion of a random input, e.g. when considering Karhunen-Loève expansions of random input fields, see Section 4.3.1 for an example. We do not make a distinction in the following, referring interchangeably to $\boldsymbol{y}$ as the *uncertain input* or *parameter* of the model. In Section 4, we show some concrete examples of such random / parametric inputs.

We denote the mapping from a parameter $\boldsymbol{y} \in U$ to the corresponding solution by $G : U \to \mathcal{X}$, $\boldsymbol{y} \mapsto G(\boldsymbol{y})$, where $\mathcal{X}$ is a Banach space. For example, in PDE models $\mathcal{X}$ could be the function space of solutions. Often, we wish to compute a quantity of interest (QoI) depending on the parameter, which we denote by $\phi : U \to \mathcal{Y}$ where $\mathcal{Y}$ is a Banach space. In the following, we assume $\phi(\boldsymbol{y}) = q(\boldsymbol{y})$, corresponding to the computation of the expected response, and thus simply write G instead of ϕ. See Section 3.4.2 for a remark on the implementation of multiple QoIs depending on the same parameter, and Section 4.3.2 for an example. In practice, a common case is $\mathcal{Y} = \mathbb{R}^d$, where often $d = 1$.

Multilevel Discretization.

We assume now that the evaluation of the mapping G for some parametric or uncertain input is burdened with a certain error, referred to in the following as the *discretization error*. Furthermore, we require for a multilevel formulation the concept of a discretization level. Often, the discretization level has the geometric interpretation as the level of mesh refinement, however this need not be the case, e.g. when considering adaptive methods with specified tolerance that decreases with increasing level.

For a maximal level L corresponding to the most exact (finest) discretization, we index the levels by $\ell = 0, \dots, L$. Since we wish to obtain a very general formulation, we allow the sample and solution spaces to be different on each level; the relevance of this generality can be seen in the description of the implementation in Section 3 below. On each level, we thus assume the input parameter to be $\boldsymbol{y}_\ell \in \mathcal{S}_\ell$, where the spaces of samples $\mathcal{S}_\ell$ fulfill $\mathcal{S}_0 \subseteq \mathcal{S}_1 \subseteq \dots \subseteq \mathcal{S}_L \subseteq U$. We further assume that a *sample restriction operation* mapping samples from a fine level onto a coarser level is given by the mapping $R_\ell : \mathcal{S}_\ell \to \mathcal{S}_{\ell-1}$ for $\ell = 1, \dots, L$. We denote the solution space on level ℓ by $\mathcal{X}_\ell$ with $\mathcal{X}_0 \subseteq \mathcal{X}_1 \subseteq \dots \subseteq \mathcal{X}_L \subset \mathcal{X}$, and require a *solution prolongation mapping* $P_\ell : \mathcal{X}_{\ell-1} \to \mathcal{X}_\ell$ for $\ell = 1, \dots, L$. This prolongation mapping will allow us to combine approximations on different levels to obtain a result on the finest level.

For each level $\ell = 0, \dots, L$, we now assume that the model can be discretized to yield an approximation $G \approx G_\ell : \mathcal{S}_\ell \to \mathcal{X}_\ell \subset \mathcal{X}$. The idea is to fix the maximal level L based on some given error tolerance, and write the approximation G_L as a telescopic sum, where for the moment we assume $R_\ell =$ id (the identity) for all ℓ,

$$G_L(\boldsymbol{y}) = \widetilde{P}_0 G_0(\boldsymbol{y}) + \sum_{\ell=1}^{L} \widetilde{P}_\ell (G_\ell(\boldsymbol{y}) - P_\ell G_{\ell-1}(\boldsymbol{y})), \qquad (1)$$

with $\widetilde{P}_\ell = P_L \circ \ldots \circ P_{\ell+1}$ for $\ell < L$ and $\widetilde{P}_L = \text{id}$. Denoting the mathematical expectation over the uncertain inputs $\boldsymbol{y}$ by $\mathbb{E}$, we obtain by linearity of the expectation and independence of P_ℓ on the samples (for $\ell = 0, \ldots, L$) the multilevel formulation

$$\mathbb{E}[G_L] = \widetilde{P}_0 \mathbb{E}[G_0] + \sum_{\ell=1}^{L} \widetilde{P}_\ell \mathbb{E}[G_\ell - P_\ell G_{\ell-1}]. \qquad (2)$$

Discretization Error.

The formulation (2) above involves the approximation of the solution on various levels, resulting in discretization errors. In the following, we denote by δ_ℓ an upper bound on the discretization error on level ℓ, i.e. assume there exist constants $\beta, C > 0$ independent of ℓ such that for all $\boldsymbol{y} \in \mathcal{S}_\ell$ holds that

$$\|G_\ell(\boldsymbol{y}) - G(\boldsymbol{y})\|_{\mathcal{X}} \leq C\delta_\ell, \quad \delta_\ell = 2^{-\beta\ell}. \qquad (3)$$

Work Model.

Since the goal of multilevel methods is to obtain a better convergence rate of the error vs. work, we need a model for the work per sample on each level $\ell = 0, \ldots, L$. In the following, we assume a model often encountered in settings involving geometric refinement of e.g. FEM meshes, namely that the work on level ℓ is given for $\gamma > 0$ by

$$w_\ell = \mathcal{O}(2^{\gamma\ell}). \qquad (4)$$

2.2 Sampling Methods

In this section we briefly review some basics of Monte Carlo and quasi-Monte Carlo methods for approximating integrals, which we will apply to the approximation of expectations. We refer to standard literature [11, 28, 29] for details to allow more focus on the description of the ensuing implementation.

To allow a unified treatment of MC and QMC, we consider the problem of integrating a function $f : [0,1]^s \to \mathbb{R}$ with respect to the Lebesgue measure, i.e. we assume a uniformly distributed input parameter $\boldsymbol{y} \sim \mathcal{U}([0,1]^s)$. If the input under consideration is a random variable distributed according to some other distribution, an integral over $[0,1]^s$ can be obtained by transforming with the inverse of the corresponding multivariate cumulative distribution function. We make this assumption only for brevity of exposition – the implementation supports arbitrary distributions in the MC case, so long as a function to sample from the distribution is provided.

2.2.1 Monte Carlo

We denote by $\{\boldsymbol{y}^{(i)} : i = 1, \ldots, M\}$ a collection of M i.i.d. realizations of a random variable $\boldsymbol{y} \sim \mathcal{U}(U)$, $U = [0,1]^s$. The Monte Carlo quadrature rule based on such a collection of M samples applied to the function f is then given by

$$\mathcal{Q}_M[f] := \frac{1}{M} \sum_{i=1}^{M} f(\boldsymbol{y}^{(i)}). \qquad (5)$$

Since $\mathcal{Q}_M[f]$ is simply a sum of M evaluations of f at random points, it too is a random variable – computing (5) multiple times will yield different values. However, it can easily be shown that the estimator (5) is unbiased, i.e. its expectation is equal to the exact value $\mathbb{E}[f]$.

In the context of the Monte Carlo method, we consider the $L^2(U)$ error, sometimes called the root-mean-square error (RMSE) which is given for a random variable X by $RMSE(X) = \sqrt{\text{Var}(X)}$. For the estimator (5) applied to a function $f \in L^2(U)$, we have $RMSE(\mathcal{Q}_M[f]) = \sqrt{\text{Var}[f]M^{-1}}$.

2.2.2 Quasi-Monte Carlo

In contrast to Monte Carlo methods, quasi-Monte Carlo (QMC) methods are not based on random sampling but on deterministically chosen quadrature points. Like MC rules, QMC rules are equal-weight rules of the form (5), where now the points $\boldsymbol{y}^{(i)} \in [0,1]^s$ are deterministically chosen, and $\mathcal{P}_M = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(M)}\}$ denotes a point set containing M points. In the following, we denote QMC quadrature with point set $\mathcal{P}_M$ applied to the function f by $\mathcal{Q}_{\mathcal{P}_M}[f]$.

Various choices for the point set $\mathcal{P}_M$ exist, and we refer to [11, 28, 29] for an overview. We consider in this paper so-called *interlaced polynomial lattice rules* [9, 12, 20] which allow convergence rates of higher order for certain integrands, thereby enabling high convergence rates in the multilevel methods. As opposed to MC rules, QMC rules are deterministic. Thus, the notion of RMSE error is replaced by the (deterministic) integration error

$$err(\mathcal{Q}_{\mathcal{P}_M}[f]) = |\mathcal{Q}_{\mathcal{P}_M}[f] - \mathbb{E}[f]|, \qquad (6)$$

where the absolute value is to be replaced by a suitable norm if the function f is not real-valued.

Sampling Error.

We assume in the following for both the Monte Carlo and quasi-Monte Carlo case that the error (either RMSE or QMC error (6)) is bounded for $C_f, \sigma > 0$ by

$$err(\mathcal{Q}_{\mathcal{P}_M}[f]) \leq C_f M^{-\sigma}, \qquad (7)$$

where σ depends on the integrand and on the choice of the point set, and C_f may depend on f. For Monte Carlo, we have $\sigma = 1/2$ for square-integrable integrand functions f.

Polynomial Lattice Rules.

Before considering *interlaced* polynomial lattice rules, we briefly introduce some necessary notation for general polynomial lattice rules. Denote by $\mathbb{Z}_b[x]$ the polynomials over the ring $\mathbb{Z}_b = (\{0, 1, \ldots, b-1\}, +_b, \cdot_b)$, where $+_b$ and $\cdot_b$ denote addition and multiplication of integers modulo a prime b. Let $G_{b,m} = \{p \in \mathbb{Z}_b[x] : p \neq 0, \deg(p) < m\}$ denote the set of nonzero polynomials of degree less than m, and assume given an irreducible polynomial $P(x)$ with $\deg(P) = m$. For a dimension $s \in \mathbb{N}$, a *generating vector* is a vector $\boldsymbol{q} \in G_{b,m}^s$, $\boldsymbol{q} = (q_1(x), \ldots, q_s(x))$ that will define $N = b^m$ quadrature points in $[0,1)^s$.

For an integer $n \in \mathbb{Z}_b$ and base $b \in \mathbb{N}$, $b > 1$, we define the polynomial $p_n(x) = n_0 + n_1 x + \ldots + n_{m-1}x^{m-1}$ with coefficients $n_0, \ldots, n_{m-1}$ such that $p_n(b) = n$. We define the mapping $v_m : \mathbb{Z}_b((x^{-1})) \to [0,1)$ which maps formal Laurent series of the form $\sum_{i \geq 1} \rho_i x^{-i}$ to the half-open unit interval by evaluating at $x = b$ and truncating after m terms,

$$v_m(p) = \sum_{i=1}^{m} \rho_i b^{-i}, \quad p(x) = \rho_1 x^{-1} + \rho_2 x^{-2} + \dots. \tag{8}$$

Then, the elements of the polynomial lattice point set $\mathcal{P}_{b^m}(\boldsymbol{q}) = \{\boldsymbol{y}^{(0)}, \dots, \boldsymbol{y}^{(b^m-1)}\} \subset [0,1)^s$ are given by

$$\boldsymbol{y}^{(n)} = \left(v_m\left(\frac{p_n q_1}{P}\right), \dots, v_m\left(\frac{p_n q_s}{P}\right)\right). \tag{9}$$

Interlaced Polynomial Lattice Rules.

Instead of considering a generating vector in s dimensions, we consider one in αs dimensions, for some *interlacing parameter* $\alpha \in \mathbb{N}, \alpha \geq 2$. We define the interlacing function $\mathcal{D}_\alpha(x_1, \dots, x_\alpha)$ where the arguments have b-adic expansion $x_i = \xi_{i,1} b^{-1} + \xi_{i,2} b^{-2} + \dots$ for $i = 1, \dots, \alpha$ by

$$\mathcal{D}_\alpha(x_1, \dots, x_\alpha) = \sum_{a=1}^{\infty} \sum_{k=1}^{\alpha} \xi_{k,a} b^{-k-(a-1)\alpha}. \tag{10}$$

For $j = 1, \dots, s$ and $n = 0, \dots, b^m - 1$, the j-th component of the n-th point $\boldsymbol{y}^{(n)}$ of the point set $\mathcal{P}_{b^m}(\boldsymbol{q})$ is obtained by interlacing blocks of α points,

$$y_j^{(n)} = \mathcal{D}_\alpha\left(v_m\left(\frac{p_n q_{(j-1)\alpha+1}}{P}\right), \dots, v_m\left(\frac{p_n q_{j\alpha}}{P}\right)\right). \tag{11}$$

Generating vectors can be efficiently obtained for various classes of integrands by the Component-by-Component (CBC) algorithm [32, 33, 34, 9, 16], which we will not focus on here. We refer to Section 3.4.3 for details on the provided generating vectors and how to use them in an implementation.

We now state a derivative bound on the function f, which we assume to hold in the following. Let $\boldsymbol{\nu}$ be a finitely supported multi-index and $E \cup E^c = \mathbb{N}$, and denote the restriction $\boldsymbol{\nu}_A = (\nu_j)_{j \in A}$ for any set $A \subset \mathbb{N}$. Then, denoting $|\boldsymbol{\nu}| = \sum_{j \geq 1} \nu_j$ and $\boldsymbol{\nu}! = \prod_{j \geq 1} \nu_j!$, we assume f to fulfill

$$\sup_{\boldsymbol{y} \in U} |(\partial_{\boldsymbol{y}}^{\nu} f)(\boldsymbol{y})| \leq C \boldsymbol{\nu}_E! \prod_{j \in E} \beta_j^{\nu_j} \times |\boldsymbol{\nu}_{E^c}|! \prod_{j \in E^c} \beta_j^{\nu_j}, \tag{12}$$

for a $C > 0$ and a sequence $\boldsymbol{\beta} \in \ell^p(\mathbb{N})$ for some $0 < p < 1$. It was shown in [9] that for integrand functions f satisfying a derivative bound of the form (12), such IPL rules converge with rate $N^{-1/p}$ independent of the truncation dimension s, and can be constructed with work scaling at most quadratically in s. For $p = 1/2$ and $\alpha = 2$, this gives rate N^{-2}, which is much better than the rate $N^{-1/2}$ of MC methods.

A further advantage of IPL rules are that they are formulated in a worst-case framework, not a mean-square framework like Monte Carlo. Thus, no randomization is needed and all error bounds are for the worst case. Classical QMC rules based on low-discrepancy sequences are also based on a worst-case analysis, however they are intrinsically limited for large s since their integration error depends on the dimension. There exist randomized lattice rules which achieve dimension-independent convergence rates, however these are limited to rate N^{-1} and again rely on an L^2 setting, see e.g. [21, 30].

2.3 Single-Level Algorithm

In a single-level context, we consider only one discretization level, denoted L, which is chosen to fulfill a certain prescribed error tolerance. Following the notation from (3) above, the discretization error is assumed to converge like $\delta_L = \mathcal{O}(2^{-\beta L})$ for some problem-dependent parameter β.

Error vs. Work.

For single-level methods, we balance the asymptotic sampling and discretization errors and consider the convergence with respect to the work. Assuming the work model $w_L = 2^{\gamma L}$ as in (4) for a problem dependent constant γ, the total work is given by

$$W_{tot}^{SL} = M w_L, \tag{13}$$

since for each of the M samples we do w_L units of work.

We now equilibrate the sampling error $err(\mathcal{Q}_{\mathcal{P}_M}[f])$ (assumed to fulfill (7)) and the discretization error δ_l to determine the number of required samples $M = \delta_L^{-1/\sigma} = 2^{\beta L/\sigma}$. This yields the total work $W = \mathcal{O}(M w_L) = \mathcal{O}(\delta_L^{-(\gamma\sigma+\beta)/(\sigma\beta)})$. Since the error is proportional to δ_L, we obtain an error vs. work relationship of

$$err \sim work^{-r}, \quad r = \frac{\sigma\beta}{\gamma\sigma + \beta}. \tag{14}$$

For various values of β, γ, σ, the resulting theoretical convergence rates of the single-level algorithm are listed in Table 1.

2.4 Multilevel Algorithm

Our goal here is to exploit the formulation (2) to obtain a method with a better convergence rate of the error vs. the work than in the single-level case above. The idea is to approximate the expectations on each level with a different quadrature rule, using many points on coarse levels (where each evaluation of the integrand is cheap) and few points on finer levels. Since the integrand on all levels except the first is a difference of two approximations with varying accuracies, we can hope that the magnitude of the integrand, and thus its quadrature error, will be small. When using a randomized method, we need that the variance of the differences decrease in size for increasing level.

For notational brevity, we assume $\mathcal{S}_\ell = \mathcal{S}$ and $\mathcal{X}_\ell = \mathcal{X}$ for $\ell = 0, \dots, L$, allowing us to omit the restrictions R_ℓ and prolongations P_ℓ in (2). Then, denoting by $\mathcal{Q}_M$ the MC or QMC sampling method with M samples and defining $G_{-1} \equiv 0$, we can write the multilevel (Q)MC estimate as

$$\mathcal{Q}^L[G_L] = \sum_{\ell=0}^{L} \mathcal{Q}_{M_\ell}[G_\ell - G_{\ell-1}], \tag{15}$$

for a given distribution of samples per level $M_0, \dots, M_L$. The total error is then given for $C_G > 0$ up to log factors by

$$\begin{aligned} \left\|\mathbb{E}[G] - \mathcal{Q}^L[G_L]\right\|_{\mathcal{X}} &\leq \|\mathbb{E}[G - G_L]\|_{\mathcal{X}} + \left\|(\mathbb{E} - \mathcal{Q}^L)[G_L]\right\|_{\mathcal{X}} \\ &\leq \delta_L + \sum_{\ell=0}^{L} \|(\mathbb{E} - \mathcal{Q}_{M_\ell})[G_\ell - G_{\ell-1}]\|_{\mathcal{X}} \\ &\leq \delta_L + C_G \sum_{\ell=0}^{L} \delta_\ell M_\ell^{-\sigma}. \end{aligned} \tag{16}$$

The total work is given by

$$W_{tot}^{ML} = \sum_{\ell=0}^{L} M_\ell w_\ell. \tag{17}$$

Before a convergence rate can be derived, a choice of the number of samples per level M_ℓ must be made. This can be found by solving an optimization problem which minimizes the error for fixed work, using a Lagrange multiplier argument, see [10, 15, 17]. In this case, this procedure yields the choice

$$M_\ell = \left\lceil M_0 2^{\frac{-(\gamma+\beta)\ell}{\sigma+1}} \right\rceil \quad \text{for } \ell = 1, \ldots, L, \tag{18}$$

with $M_0 = \lceil E^{1/\sigma} 2^{\frac{\beta L}{\sigma}} \rceil$ for $E = \sum_{\ell=0}^{L} 2^{((\gamma\sigma-\beta)\ell)/(\sigma+1)}$. Using this M_ℓ, an explicit derivation of the error vs. work behavior can be conducted to prove the exact rates. We omit such derivations for brevity, referring instead to [17] for MLMC and [10] for the HoQMC case. In the following, we estimate the error vs. work convergence rate by computing the error (16) and the work W_{tot} for M_ℓ as in (18) and various L, and estimating the rate by a linear least squares fit. The resulting predicted convergence rates are listed in Table 1 for various choices of the parameters β, γ, σ.

β	σ	$\gamma = 1$		$\gamma = 2$	
		SL	**ML**	**SL**	**ML**
1	0.5	0.3333	0.4315	0.25	0.3707
1	1	0.5	0.6825	0.3333	0.4678
1	2	0.6667	0.8641	0.4	0.4915
1	3	0.75	0.9186	0.4286	0.4971
1	4	0.8	0.9361	0.4444	0.4979
2	0.5	0.4	0.4938	0.3333	0.4844
2	1	0.6667	0.9360	0.5	0.8115
2	2	1	1.4822	0.6667	0.9685
2	3	1.2	1.6971	0.75	0.9891
2	4	1.3333	1.7982	0.8	0.9939
4	0.5	0.4444	0.4999	0.4	0.4998
4	1	0.8	0.9980	0.6667	0.9917
4	2	1.3333	1.9368	1	1.7032
4	3	1.7143	2.6507	1.2	1.9140
4	4	2	3.0941	1.3333	1.9651

Table 1: Convergence rate of error with respect to work for single-level (SL) and multilevel (ML) methods, for various combinations of the parameters β, σ and γ, see (3), (7) and (4), respectively. Note that the case $\sigma = 0.5$ corresponds to the MC method, when considering the L^2 error. The SL rate is given by (14) and the ML rate was obtained by a linear least-squares fit to the log(error) vs. log(work) graph for $L = 0, \ldots, L_{max} = 8$.

3. IMPLEMENTATION

In this section, we present the open-source `gMLQMC` library for applying multilevel (quasi) Monte Carlo to a general parametric model, which is available at `https://gitlab.math.ethz.ch/gantnerr/gMLQMC`. The library is designed in a generic way, allowing arbitrary user-defined types to be used, provided they fulfill some natural conditions mentioned below. Additionally, it allows arbitrary parallelization strategies to be specified, providing an extensible high-performance implementation.

3.1 Assumptions on Data Types

We write the estimator from (15) using the restriction and prolongation mappings R_ℓ and P_ℓ defined in Section 2.1 above, which may be required depending on the application at hand. For example, if the returned quantity is a coefficient vector of a finite element basis on a certain level, it will need to be prolongated to a finer level before the difference can be computed. The restrictions $R_\ell : \mathcal{S}_\ell \to \mathcal{S}_{\ell-1}$ map *samples* $\boldsymbol{y}$ from level ℓ to level $\ell - 1$, and the prolongations $P_\ell : \mathcal{X}_{\ell-1} \to \mathcal{X}_\ell$ map *solutions* $G_{\ell-1}(\boldsymbol{y})$ from level $\ell - 1$ to level ℓ. The estimator is

$$\mathcal{Q}^L[G_L] = \sum_{\ell=0}^{L} \widetilde{P}_\ell \, \mathcal{Q}_{M_\ell}[G_\ell - (P_\ell \circ G_{\ell-1} \circ R_\ell)], \tag{19}$$

where we define $G_{-1} \equiv 0$ and $\widetilde{P}_\ell = P_L \circ \ldots \circ P_{\ell+1}$.

We assume the spaces $\mathcal{X}_\ell$ to be represented by the same data type `X` for all $\ell = 0, \ldots, L$, where the only condition we require on this type is that vector space operations are defined. In other words, for `x`,`y` instances of `X` and a scalar value `a` that is convertible to `double`, the expressions `x+y` and `a*x` must compile. For more details, see the `VectorSpace` concept in the code documentation.

For the samples, we again assume the spaces $\mathcal{S}_\ell$ to be represented by the same type `S` for all levels, however we do not require any conditions on the form of `S`. This is possible since the user must specify both the function generating the samples and the function mapping samples to solutions, and thus has complete control over the sample type `S`. The default sample restriction operator is implemented generically as the identity.

We now describe how the model $G_\ell(\boldsymbol{y})$, the restrictions and prolongations, and the function generating the samples $\boldsymbol{y}$ are specified in this framework.

3.2 API

The `gMLQMC` package allows specification of an integrand function G_ℓ as any callable (a function, class with call operator, lambda or function pointer) accepting as the first argument a sample of the type `S` and as second argument the level `l` as an integer. The return type is the arbitrary type `X` representing the spaces $\mathcal{X}_\ell$. Some possibilities of specifying integrands are given in Listing 1.

```
// function version
X integrand1(S s, int l)
{ return solver(s); }

// class instance version
struct my_integrand {
    X operator()(S s, int l)
    { return solver(s); }
} integrand2; // directly instantiate

// lambda version
auto integrand3 =
    [](S s, int l){return solver(s);};
```

Listing 1: Examples of different ways to specify a serial integrand function G_ℓ.

To generate samples, the user must specify a function taking two arguments: the level `l` as an integer, and the sample index `n`, and returning samples of type `S` to be passed to the integrand on levels ℓ and $\ell - 1$. To be consistent with the integrand function, the sample type must of course be convertible to the type of the first argument of the user-specified integrand function. For convenience, various samplers are implemented in the framework, see Section 3.4.4 for an overview.

Restriction and Prolongation.

The restriction of a sample from level ℓ to level $\ell - 1$ can be realized by specifying a function of the form `re-`

`strict_sample(S&& s, int l) -> S`, which is by default simply the identity. For the prolongation of the result from a coarse level $\ell-1$ to a fine level ℓ, a function of the form `project_solution(X& x, int l) -> X` must be given, where `X` is the user-defined type of the result. By default it is simply the identity, which suffices for e.g. scalar-valued integrands.

Executing MLQMC.

Once the relevant functions are present, the serial multilevel method can be executed with a call to the function `gMLQMC::MLQMC(integrand,sampler,Ml)`, where `Ml` is a vector of $L+1$ sample numbers $M_0, \dots, M_L$. This vector specifies both M_ℓ and the index of the maximal level L, which is determined based on its length. If restriction and prolongation operations were defined, they must be specified to the underlying class instance. This can be done as in Listing 2.

```
// Specify sample restriction operation
// for sample type 'sam_t', e.g.
auto R = [](sam_t&& sam, int l){ return sam[0]; };
// Specify solution prolongation operation
// for solution type 'sol_t', e.g.
auto P = [&mesh](sol_t& sol, int l)
    { return mesh.prolong(sol,l); };
// Instantiate ML simulation class explicitly.
gMLQMC::gMLQMC_simulation<int_t,sam_t,serial>
    gmlqmc(integrand, mysampler);
// Set restriction and prolongation operations.
// Note that the specification of options can be
// chained, and the ordering can be arbitrary.
gmlqmc.restrict_sample(R)
      .prolongate_solution(P);
// Execute serially. Ml gives the sample numbers.
auto result = gmlqmc.run(Ml);
```

Listing 2: Example of call to gMLQMC with specification of restriction and prolongation operations.

Choice of β, γ, σ and M_ℓ.

The file `gMLQMC/tools/standard_Ml.hpp` contains a routine for computing M_ℓ according to (18), given the values of β, γ, σ. The values β and γ are the convergence rate of the discretization error and the scaling of the work complexity, respectively, and can often be derived from properties of the integrand. If not known a-priori, they can be estimated in a precomputation step by a fit to the graph of the discretization error or computational work vs. the level ℓ. The choice of sampling method fixes σ, see e.g. Section 4.1.3.

3.3 Parallelization

We use MPI for parallelization, which allows execution on the largest parallel computers currently available. Moreover, the well-behaved scaling properties of this software will allow its use in the context of exascale computing. It should be mentioned that MLMC methods have been shown to be intrinsically fault-tolerant [35], further justifying their possible future use in this context.

In order to obtain a parallel implementation that allows a similar generality with respect to the involved types, we rely on the `boost::mpi` library, which only requires that a type is "serializable" [1] in order to send or receive it between nodes. In short, the user-defined type `X` must either contain a member function `serialize`, or a global `serialize` function must be available. Many types are already supported by Boost or gMLQMC, in particular all basic types, `std::vector<T>` if `T` is serializable, the types of the Eigen matrix library [22], and many more.

The gMLQMC library supports both parallel evaluation of a "serial integrand", by which we mean a function that itself does not require more than one processor to run, as well as parallel evaluation of a "parallel integrand", leading to nested parallelism. For parallel integrands, the function signature introduced above with two parameters (the sample and the level) does not suffice. For such functions to be compatible with this library, they must accept a third argument, the MPI communicator [2] representing the group of processors the function is allowed to use. Depending on its signature (which is determined at compile-time), the third argument is automatically passed. See Section 4.3.3 for an example.

We denote by $\{P_\ell\}_{\ell=0}^L$ the number of workers per level, which are responsible for computing $G_\ell - G_{\ell-1}$ for $\ell \geq 1$ and G_0 for $\ell = 0$, and by $\{D_\ell\}_{\ell=0}^L$ the number of CPUs per worker on each level, which is only relevant for parallel integrands. Vectors containing these values can be passed to the `MLQMC` function, and are then passed to the specified strategy.

Parallelization strategies are located in the `gMLQMC/strategy` directory, and are specified as a template argument to the `MLQMC` function. The `gMLQMC::strategy::full` strategy implements full parallelization, with $N_{CPU} = \sum_{\ell=0}^L P_\ell D_\ell$ total CPUs. It uses the method from [43] and assumes P_ℓ and D_ℓ given for $\ell = 0, \dots, L$. For a working example, see the file `examples/strategy_full.cpp` in the gMLQMC library folder. A serial strategy for small simulations and testing purposes is given by `gMLQMC::strategy::serial`.

3.4 User Guide

gMLQMC is a header-only library, which means that no library files need to be compiled and installed prior to use. To use the library, simply extract the provided archive file to a known location, and add the `gMLQMC/include` subfolder to the include path when compiling. In Table 2, a list of software dependencies is given.

3.4.1 Compiling Tests and Examples

The two subfolders `examples` and `test` of the code archive contain some example programs and test cases, respectively. The CMake build system [27] is used to automatically detect dependencies and their include and library locations, and to generate makefiles for compiling the code.

The workflow is as follows: first, a build directory should be created, from which the `cmake` command should be called with the path to the root of gMLQMC. For example, if you are currently in the root of the code archive, execute:

```
mkdir build
cd build
cmake ..
```

Then, in the build folder, execute the following to compile and run the serial and parallel tests:

```
make serial_testrun
make parallel_testrun
```

and the following for the compiling all examples and listing the built executables:

[1]For a formal specification of the requirements, see the documentation at `www.boost.org/libs/serialization/doc/`.

[2]Note that the third argument can be either a pure MPI communicator or a `boost::mpi::communicator`, since they are convertible into each other.

Library	Description
Boost:: Serialization	This library is required to allow generic serialization of objects, which is used by `boost::mpi` for communication operations.
Boost::MPI	This library provides a C++ wrapper for the MPI interface, and simplifies many of the required operations. This library must be consistent with the version of MPI used. The user-specified functions may use any MPI commands, and are not required to use Boost::MPI.
MPI	An implementation of MPI must be available. `gMLQMC` has been tested with `openMPI` and `mpich`.
`catch.hpp`*	This header-only testing library is required for executing the unit tests. `https://github.com/philsquared/Catch`.
`jsoncons`*†	This is a header-only implementation of the JSON file format, used for reading generating vectors `https://github.com/danielaparker/jsoncons`.
NTL†	If IPL rules are to be used, the NTL (Number Theory Library) [40] must be available. For CMake to find the library, set the environment variable `NTL_DIR=/path/to/ntl`.
GMP†	This is a dependency of NTL. For CMake to find the library, set the environment variable `GMP_DIR=/path/to/gmp`.

Table 2: Dependencies of `gMLQMC`. Those marked * are included in the code archive, those with † are only required if IPL rules are used. For serial execution, MPI and Boost libraries can be omitted.

Sampler	Description
Uniform	Random sampling from the uniform distribution $\mathcal{U}([a,b])$. In: `gMLQMC/samplers/uniform.hpp`
Normal	Random sampling from the normal distribution $\mathcal{N}(\mu,\Gamma)$, where $\mu \in \mathbb{R}^s$ and $\Gamma \in \mathbb{R}^{s\times s}_{sym}$. In: `gMLQMC/samplers/normal.hpp`
Halton	Implements the Halton sequence in s dimensions, where by default the first s primes are used as the bases. In: `gMLQMC/samplers/halton.hpp`
IPL	Evaluation of interlaced polynomial lattice rules, initialized with a lattice obtainable as mentioned in Section 3.4.3 above. In: `gMLQMC/samplers/IPL.hpp`

Table 3: Samplers provided with `gMLQMC`.

```
make examples
ls examples/
```

Alternatively, single examples can be executed by calling `make example_name` where `example_name` is the name of the .cpp file in the `examples` folder without the extension.

3.4.2 Multiple Quantities of Interest

In the presentation above, we consider only the approximation of the expectation of G, not of any higher moments or other QoI functionals ϕ. We give here a straightforward method for implementing the computation of further quantities of interest, without significantly complicating the implementation. We assume the number of samples in the multilevel method to be previously fixed, e.g. by techniques mentioned in [18, Sec. 2.5].

If, for example, G is real-valued and $\phi : U \to \mathbb{R}^2$ with $\phi(\boldsymbol{y}) = (G(\boldsymbol{y}), (G(\boldsymbol{y}))^2)$, the variance $\mathrm{Var}[G] = \mathbb{E}[G^2] - \mathbb{E}[G]^2$ can be computed as $Y_2 - Y_1^2$, where $Y = \mathbb{E}[\phi(\boldsymbol{y})]$. Of course, ϕ can be implemented such that G is evaluated only once for each $\boldsymbol{y}$. Further applications of this technique include Bayesian inversion, see Section 4.3.2.

More generally, if we denote $\phi : U \to \mathcal{Y}$, a representation of $\mathcal{Y}$ must be implemented in the form of a type `Y`, and vector space operations required by the `gMLQMC` framework must be specified (see Section 3.1). This can be tedious and time-consuming, and has to be redone every time a new variation of the QoI is considered.

To alleviate this, we provide a class template called `direct_product`, which, given an arbitrary number of template parameters, constructs a type representing the direct product of the given types. If the individual types support vector space operations (i.e. expressions involving the addition of two instances or multiplication with a `double` compile), the resulting type will automatically support vector space operations. An example is given in Listing 3.

3.4.3 Generating Vector Repository

We include in the subdirectory `genvecs` a selection of generating vectors for IPL rules suitable for high-dimensional numerical integration of functions of the type considered in [9, 16], for product and SPOD-type weights. For an example of HOQMC quadrature applied to some of the functions considered in [16], see `examples/IPL_quadrature.cpp`.

IPL rules involve various parameters which affect their performance. We do not discuss details of this here, referencing instead a repository of generating vectors for IPL rules located at `http://www.sam.math.ethz.ch/HOQMC`, which contains generating vectors for various canonical situations.

3.4.4 Available Samplers

Some common samplers are provided with the `gMLQMC` library, and are listed in Table 3. They assume the sample type `S` to be `std::vector<double>`, which is a common choice. For more details on how to use custom samplers, see the code documentation. See `gMLQMC/samplers/random_base.hpp` for a general base class for random samplers.

4. EXAMPLES

4.1 Parametric Integration

This example is inspired by the original 2001 paper by Heinrich, which first developed MLMC in the context of the computation of parametric integrals [23]. Our goal in this example is to give a simple illustration of the ML(Q)MC algorithm and the resulting expected and measured convergence rates. We wish to stress that in terms of convergence rates of the discretization error, this simple example models the same behavior as more complex PDE-based models relevant in practical applications.

We consider the problem of computing the following expectation, where we assume the random parameter $\boldsymbol{y}$ to follow a uniform distribution on the s-dimensional unit cube,

$$\mathbb{E}\left[I(\boldsymbol{y})\right] = \mathbb{E}\left[\int_0^1 f(x,\boldsymbol{y})\,\mathrm{d}x\right] = \int_{[0,1]^{s+1}} f(x,\boldsymbol{y})\,\mathrm{d}x\,\mathrm{d}\boldsymbol{y}.$$

In the following, we consider the function

$$f(x,\boldsymbol{y}) = \left(u_0 + \sum_{j=1}^{s}(2y_j-1)\psi_j(x)\right)^{-1},$$

where we define the basis functions by $\psi_j(x) = j^{-2}\sin(j\pi x)$ for $j = 1,\ldots,s$ and use $u_0 \equiv \pi$. We seek to compute the expectation of $I(\boldsymbol{y})$, and now introduce a discretization of I in the form of one-dimensional quadrature over x, where we increase the number of quadrature points as we go to finer levels. Note that the use of the term ‘quadrature’ may seem confusing; we use ‘quadrature’ solely for the integral over x and speak of ‘sampling’ the integrand $I(\boldsymbol{y})$ at different realizations of the parameter $\boldsymbol{y}$ (by an MC or QMC method).

4.1.1 Discretization Methods

We consider multiple levels of discretization, where on discretization level ℓ we use $N_\ell = 2^{\ell_0+\ell}+1$ quadrature points. In order to be able to examine different combinations of convergence rate and discretization error $\delta_\ell = \mathcal{O}(2^{-\beta\ell})$, we consider three different quadrature rules:

1. Van der Corput sequence: $\delta_\ell = \mathcal{O}(N_\ell^{-1+\varepsilon})$ [13, 3.4.2],
2. Trapezoidal rule: $\delta_\ell = \mathcal{O}(N_\ell^{-2})$,
3. Simpson's rule: $\delta_\ell = \mathcal{O}(N_\ell^{-4})$.

We thus consider in the following the values $\beta \in \{1,2,4\}$. The *coarsest level* $\ell = 0$ has the fewest quadrature points N_0, and thus the greatest quadrature (or discretization) error for a given realization of $\boldsymbol{y}$. On the *finest level* with $\ell = L$, we have $N_L \gg N_0$ quadrature points, giving us a very good (but expensive) approximation of $I(\boldsymbol{y})$, for any $\boldsymbol{y}$. The work is assumed to be proportional to the number of function evaluations, so we set $w_\ell = N_\ell = \mathcal{O}(2^\ell)$, giving $\gamma = 1$ in (4).

4.1.2 Properties of the Integrand

The form of this basis is crucial to the applicability of HOQMC methods: the norm $\|\psi_j\|_{L^\infty([0,1])}$ decreases like j^{-2} with respect to the coordinate index j. This means, intuitively, that the integration variables y_j become successively less important (asymptotically) as the dimension increases, with a known rate (which is not always available).

Combined with a derivative bound of the form (12), the property that in the infinite-dimensional case the sequence $(\|\psi_j\|_{L^\infty})_{j\geq 1} \in \ell^p(\mathbb{N})$ for a $0 < p < 1$ is called *sparsity* [6, 11, 37], and allows such high dimensional integrals to be approximated by an M-point IPL rule with rate $M^{-1/p}$ [9, Thm. 3.1]. Here, this property holds for $p > 1/2$, yielding a convergence rate of the QMC approximation of M^{-2}. In the notation of MLQMC developed above, this corresponds to the case $\sigma = 2$. See [16] for examples of the application of QMC quadratures to integrands with similar properties, including numerical results verifying these rates. Note that for a large class of parametric operator equations with holomorphic parameter dependence, their solutions can be shown to exhibit the same type of properties, allowing these higher-order QMC methods to be applied [8].

4.1.3 Sampling Methods

We consider the following sampling methods for approximating the expectation over $U = [0,1]^s$, which have an error convergence rate of $err(\mathcal{Q}_M[f]) = \mathcal{O}(M^{-\sigma})$:

a. Monte Carlo: $\sigma = 1/2$ (RMSE error),

b. Halton sequence [31, p. 29]: $\sigma = 1-\varepsilon$,

c. Interlaced polynomial lattice rule with $\alpha = 2$: $\sigma = 2$.

4.1.4 Results

To study the convergence of the error vs. work, we vary the maximal discretization level in the range $L = 0,\ldots,8$. For the single-level algorithm, we proceed as in Section 2.3, choosing the number of (Q)MC points as $M = 2^{\beta L/\sigma}$, where β and σ are as in the lists in Sections 4.1.1 and 4.1.3 above, respectively. We compute the work as in (13). In the multilevel case, we choose the number of samples per level as in (18) and compute the work with (17).

Figures 1 to 3 show the convergence rates for the different sampling methods listed above, for $s = 10$. The results correspond to the theoretical rates from Table 1. All errors were computed with respect to a reference solution on level $L = 11$ obtained using SL IPL sampling. In the MC case, $R = 20$ repetitions were used to estimate the L^2 error.

In Figure 4, the various sampling methods are directly compared, showing the vast gains possible when considering higher-order convergent discretizations, both when switching from a single-level to a multilevel formulation, and from a MC to a QMC or HoQMC method.

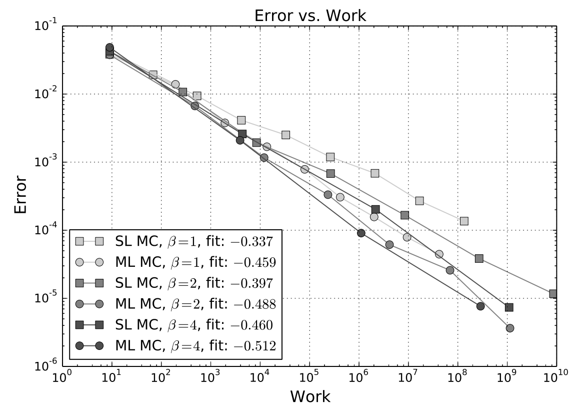

Figure 1: Convergence of L^2 error vs. work for MC sampling. The L^2 error was computed by averaging over $R = 20$ repetitions and approaches the optimal rate, i.e. $error \sim work^{-1/2}$.

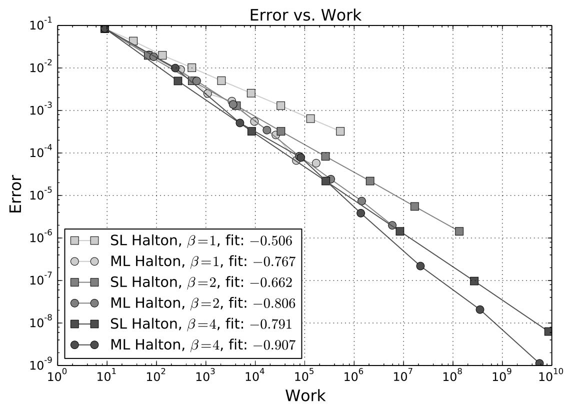

Figure 2: Convergence of error vs. work for Halton sampling, where the optimal rate is $error \sim work^{-1+\varepsilon}$.

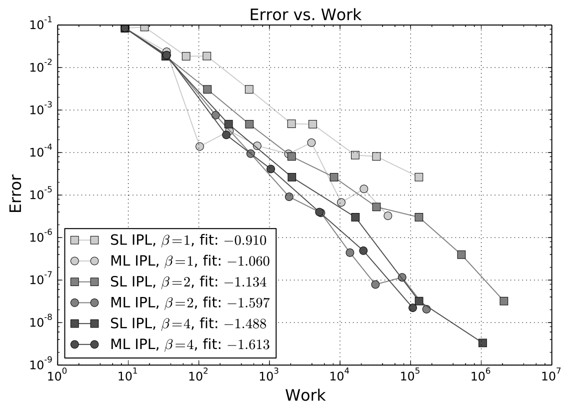

Figure 3: Convergence of error vs. work for interlaced polynomial lattice rules, where the optimal rate is $error \sim work^{-\alpha}$ for interlacing parameter $\alpha \geq 2$. Here, we use $\alpha = 2$.

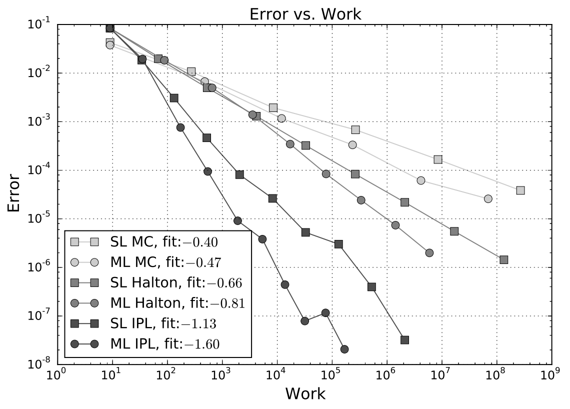

Figure 4: Convergence of error vs. work for various sampling methods, with $\beta = 2$ in the discretization error (3). For error level 10^{-5}, each ML method is approximately one order of magnitude faster than the corresponding SL method, while ML with IPL sampling is around 5 orders of magnitude faster than MLMC.

4.2 Parallel Scaling

We consider a parametric diffusion equation posed in a 2d spatial domain $D = (0,1)^2$ with parameter $\boldsymbol{y} \in [-\frac{1}{2}, \frac{1}{2}]^{\mathbb{N}}$,

$$-\nabla \cdot (u(x;\boldsymbol{y})\nabla q(x;\boldsymbol{y})) = f(x) \text{ in } D,\; q(x;\boldsymbol{y}) = 0 \text{ for } x \in \partial D,$$

where the uncertain diffusion coefficient u is parametrized by $u(x;\boldsymbol{y}) = u_0 + \sum_{j\geq 1} y_j \psi_j(x)$. For a suitable choice of u_0 and $(\psi_j)_{j\geq 1}$, chosen here as in [10], we have that the sequence $(\|\psi_j\|_{L^\infty(D)})_{j\geq 1} \in \ell^p(\mathbb{N})$ for some $p < \frac{1}{2}$, where u_0 is chosen to ensure $u(x,\boldsymbol{y}) \geq \check{u} > 0\ \forall \boldsymbol{y}$. This allows integrals of functionals of the parametric solution $q(x;\boldsymbol{y})$ to be approximated by interlaced polynomial lattice rules with higher orders of convergence [9, 10, 8]. We consider the QoI $\phi(\boldsymbol{y}) = \int_D q(x;\boldsymbol{y})\,\mathrm{d}x$ and aim to approximate $\mathbb{E}[\phi(\boldsymbol{y})]$, where we consider a uniform distribution $\boldsymbol{y} \sim \mathcal{U}(U)$. We solve this PDE with the finite element method with truncation dimension $s = 1024$, using the BETL library [24].

In this example, we wish to investigate parallel scaling properties of the `gMLQMC` library. To this end, we consider strong scaling with fixed total work and varying CPU number, as well as weak scaling where the amount of work per CPU is fixed. We choose $L = 5$, resulting in 6 levels of uniform mesh refinement and consider at least one CPU per level. For weak scaling, we increase the number of sampling points (chosen as in [10, (3.28)]) by a factor increasing with number of CPUs and average the execution time over 5 runs [26]. Table 4 shows the measured parallel efficiency for up to 12288 CPUs.

P	**Efficiency**	
	strong	weak
6	1.000	1.000
12	1.000	1.000
24	1.000	1.000
48	0.998	0.998
96	0.998	0.998
192	0.998	0.998
384	0.998	0.997
768	0.997	0.996
1536	0.997	0.994
3072	0.969	0.988
6144	0.944	0.962
12288	0.848	0.913

Table 4: Parallel efficiency for strong and weak scaling, computed with $e_{strong} = t_6/(Pt_P)$ and $e_{weak} = t_6/t_P$, where P is the number of CPUs and t_P the execution time of the call to `gMLQMC::MLQMC` with P CPUs. Note that $P = 6$ corresponds to one CPU per level.

4.3 Common Use Cases

We briefly mention various use cases that arise in practice and give examples of how the different components of the library can be used to efficiently implement them.

4.3.1 Expansions of Random Fields

Many applications involving distributed random inputs, e.g. random spatially varying coefficients of a PDE model, can be treated by parametrizing the input uncertainty in a countable basis. One such representation is the Karhunen-Loève expansion of a random field, see [38, 39, 41]. Assume the uncertain input $u \in V$ is in a separable Banach space

V with unconditional Schauder basis $\{\psi_j\}_{j\geq 1}$. Then, rescaling the ψ_j s.t. $U = [-\frac{1}{2}, \frac{1}{2}]^{\mathbb{N}}$, every $u \in V$ can be given for a $\boldsymbol{y} \in U$ as

$$u(\boldsymbol{y}) = u_0 + \sum_{j\geq 1} y_j \psi_j. \tag{20}$$

In a practical application, $u_0(x)$ represents the nominal input and $\psi_j(x)$ a basis of the fluctuations, which often decrease in magnitude like $\|\psi_j\|_{L^\infty} \sim j^{-a}$ for an $a > 1$. The sum in (20) can then be truncated at a *truncation dimension* $s < \infty$. A smaller s will require less work while giving a larger error, which naturally fits into the multilevel framework developed above.

Thus, if we consider a non-decreasing sequence $(s_\ell)_{\ell\geq 0}$ of truncation dimensions on different levels ℓ, a dimension truncation error must be considered in addition to the sampling and discretization errors considered above. See [10] for the corresponding multilevel analysis and example applications including numerical results.

In the multilevel context, we thus generate a sample on level ℓ in the form of a vector of coefficients $\boldsymbol{y}_\ell \in \mathbb{R}^{s_\ell}$ of (20), which must be projected from the fine level ℓ to the coarse one, $\ell - 1$. In this case, the sample restriction mapping is $R_\ell : \mathbb{R}^{s_\ell} \to \mathbb{R}^{s_{\ell-1}}$, given by $R_\ell(\boldsymbol{y}_\ell) = (y_1, \ldots, y_{s_{\ell-1}})$. A concrete example involving such a restriction is given in the file `examples/dimension_truncation.cpp` in the code repository.

4.3.2 Bayesian Inversion

Bayesian inversion considers the problem of determining moments of a functional $\phi(\boldsymbol{y})$, given a perturbed measurement $\delta = \mathcal{O}(G(\boldsymbol{y}^\star)) + \eta$ of the model G, where $\mathcal{O} : \mathcal{X} \to \mathbb{R}^K$, $K < \infty$ denotes an *observation operator*. The exact parameter $\boldsymbol{y}^\star$ is assumed unknown and the measurement error $\eta \sim \mathcal{N}(0, \Gamma)$ for given covariance matrix $\Gamma \in \mathbb{R}^{K\times K}_{sym}$.

Assuming a uniform distribution as the *prior distribution* on $\boldsymbol{y}$, Bayes' theorem yields a *posterior distribution*, denoted by $\boldsymbol{y}|\delta$, which incorporates the given measurement instance δ. Following [7, 41], the expectation of the QoI $\phi(\boldsymbol{y})$ wrt. the posterior given data δ can be written as (with $\|x\|^2_\Gamma := x^T\Gamma^{-1}x$)

$$\mathbb{E}^{\boldsymbol{y}|\delta}[\phi(\boldsymbol{y})] = \frac{Z'}{Z} := \frac{\int_U \phi(\boldsymbol{y}) \exp(-\frac{1}{2}\|\mathcal{O}(G(\boldsymbol{y})) - \delta\|^2_\Gamma)\,\mathrm{d}\boldsymbol{y}}{\int_U \exp(-\frac{1}{2}\|\mathcal{O}(G(\boldsymbol{y})) - \delta\|^2_\Gamma)\,\mathrm{d}\boldsymbol{y}},$$

which requires the computation of two integrals. Considering a single-level computation with fixed discretization level L, we must evaluate both integrands for each sample $\boldsymbol{y}$. We give in Listing 3 a brief example of a possible computational realization of this using the technique from Section 3.4.2, noting that in the MC case, MCMC and SMC methods [4, 5, 14, 25] outperform this approach but still converge like $N^{-1/2}$ asymptotically. The higher-order methods included here are relevant in this context and have been shown to outperform MC methods [7].

4.3.3 Nested Parallelism

One may wish to apply uncertainty quantification techniques like QMC to known problems for which parallel solvers utilizing MPI exist. In the context of the finite element method for PDEs, this could correspond to the use of mesh partitioning and distributed assembly and/or solution of the resulting linear system. Using the technique mentioned in Section 3.3, we wish to simply wrap a MLQMC method around an existing PDE solver without intrusive solver modifications.

We consider here again the parametric integration example from above, where now on level ℓ we use $D_\ell = 2^\ell$ CPUs to evaluate the quadrature rule in x. This example is implemented for a simple equidistant equal-weight quadrature rule and Halton sampling in `examples/nested.cpp`.

```
// Bayesian inversion with forward map G
// 1. redefine the integrand
auto integrand = [&](sample_t y, int level) {
    sol_t sol = G(y, level, captured_params);
    // potential is e.g. a captured lambda
    // meas is captured sol_t (the measurement)
    double theta = exp(-potential(meas, sol));
    return direct_product<sol_t,sol_t,double>(
        sol, theta*sol, theta);
};
// 2. execute SL algorithm
// sampler generates samples from prior distr.
// auto is direct_product<sol_t,sol_t,double>
auto out = gMLQMC::SLQMC(integrand,sampler,M,L);
// 3. extract prior and posterior means
sol_t prior_mean = get<0>(out);
sol_t posterior_mean = 1./get<2>(out)*get<1>(out);
```

Listing 3: Example of multiple QoIs in the context of Bayesian inverse problems, allowing computation of the prior and posterior expectations in one run. The observation operator is assumed to be implemented inside the `potential` function.

5. CONCLUSION

We consider approximations of high-dimensional integrals arising in uncertainty quantification and give a brief overview of single- and multilevel MC and QMC methods including derivations of their theoretical convergence rates. A generic, high-performance C++ framework implementing these methods was described, and some examples and computational results were presented. The results in Figure 4 show that extremely large gains are possible when using a combination of a higher-order discretization and multilevel higher-order QMC rules. This will allow the efficient solution of current problems in uncertainty quantification to much higher accuracies, or, conversely, the treatment of much larger problems than currently possible with Monte Carlo methods.

Acknowledgements.
This work was supported by CPU time from the Swiss National Supercomputing Centre (CSCS) under project ID d41, by the Swiss National Science Foundation (SNF) under project number SNF149819 and by the European Research Council (ERC) under FP7 Grant AdG247277. Some ideas were developed in collaboration with Sharan Jagathrakshakan in his master's thesis under the supervision of Prof. Peter Arbenz.

6. REFERENCES

[1] David F. Anderson and Desmond J. Higham. Multi-level Monte Carlo for continuous time Markov chains, with applications to biochemical kinetics. *SIAM Multiscale Modeling and Simulation*, 10(1):146–179, 2012.

[2] Andrea Barth, Annika Lang, and Christoph Schwab. Multilevel Monte Carlo method for parabolic stochastic partial differential equations. *BIT Numerical Mathematics*, 53(1):3–27, 2012.

[3] Andrea Barth, Christoph Schwab, and Nathaniel Zollinger. Multi-level Monte Carlo Finite Element method for elliptic PDEs with stochastic coefficients. *Numerische Mathematik*, 119(1):123–161, April 2011.

[4] Alexandros Beskos, Ajay Jasra, Kody Law, Raul Tempone, and Yan Zhou. Multilevel Sequential Monte Carlo Samplers. *ArXiv e-prints*, March 2015.

[5] Peter Bickel, Bo Li, and Thomas Bengtsson. Sharp failure rates for the bootstrap particle filter in high dimensions. In Bertrand Clarke and Subhashis Ghosal, editors, *Pushing the limits of contemporary statistics: contributions in honor of Jayanta K. Ghosh*, volume 3 of *Inst. Math. Stat. Collect.*, pages 318–329. Inst. Math. Statist., Beachwood, OH, 2008.

[6] Abdellah Chkifa, Albert Cohen, and Christoph Schwab. Breaking the curse of dimensionality in sparse polynomial approximation of parametric PDEs. *Journal de Mathématiques Pures et Appliquées*, 103(2):400 – 428, 2015.

[7] Josef Dick, Robert N. Gantner, Quoc T. Le Gia, and Christoph Schwab. Higher order Quasi-Monte Carlo integration for Bayesian Estimation. Technical Report 2016-13, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2016.

[8] Josef Dick, Quoc T. Le Gia, and Christoph Schwab. Higher order quasi–Monte Carlo integration for holomorphic, parametric operator equations. *SIAM/ASA Journal on Uncertainty Quantification*, 4(1):48–79, 2016.

[9] Josef Dick, Frances Y. Kuo, Quoc T. Le Gia, Dirk Nuyens, and Christoph Schwab. Higher order QMC Petrov-Galerkin discretization for affine parametric operator equations with random field inputs. *SIAM Journal on Numerical Analysis*, 52(6):2676–2702, 2014.

[10] Josef Dick, Frances Y. Kuo, Quoc T. Le Gia, and Christoph Schwab. Multi-level higher order QMC Galerkin discretization for affine parametric operator equations. *Report 2014-14 Seminar for Applied Mathematics, ETH Zürich*, 2014.

[11] Josef Dick, Frances Y. Kuo, and Ian H. Sloan. High-dimensional integration: the quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 2013.

[12] Josef Dick, Quoc T. Le Gia, and Christoph Schwab. Higher Order Quasi Monte-Carlo Integration in Uncertainty Quantification. In Robert M. Kirby, Martin Berzins, and Jan S. Hesthaven, editors, *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2014*, volume 106 of *Lecture Notes in Computational Science and Engineering*, pages 445–453. Springer International Publishing, 2015.

[13] Josef Dick and Friedrich Pillichshammer. *Digital nets and sequences*. Cambridge University Press, Cambridge, 2010. Discrepancy theory and quasi-Monte Carlo integration.

[14] Tim J. Dodwell, Chris Ketelsen, Robert Scheichl, and Aretha L. Teckentrup. A hierarchical multilevel Markov chain Monte Carlo algorithm with applications to uncertainty quantification in subsurface flow. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):1075–1108, 2015.

[15] Robert N. Gantner, Claudia Schillings, and Christoph Schwab. *Domain Decomposition Methods in Science and Engineering XXII*, chapter Binned Multilevel Monte Carlo for Bayesian Inverse Problems with Large Data, pages 511–519. Springer International Publishing, Cham, 2016.

[16] Robert N. Gantner and Christoph Schwab. Computational Higher-Order Quasi-Monte Carlo Integration. *Tech. Report 2014-25, Seminar for Applied Mathematics, ETH Zürich (to appear in Proc. MCQMC14)*, 2014.

[17] Michael B. Giles. Multilevel Monte Carlo path simulation. *Operations Research*, pages 1–25, 2008.

[18] Michael B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259–328, 5 2015.

[19] Michael B. Giles and Benjamin J. Waterhouse. Multilevel quasi-Monte Carlo path simulation. In *Advanced Financial Modelling*, volume 8 of *Radon Series on Computational and Applied Mathematics*, pages 165–182. Walter de Gruyter, Berlin, 2009.

[20] Takashi Goda and Josef Dick. Construction of Interlaced Scrambled Polynomial Lattice Rules of Arbitrary High Order. *Foundations of Computational Mathematics*, 15(5):1245–1278, 2014.

[21] Ivan G. Graham, Frances Y. Kuo, James A. Nichols, Robert Scheichl, Christoph Schwab, and Ian H. Sloan. Quasi-Monte Carlo finite element methods for elliptic PDEs with lognormal random coefficients. *Numerische Mathematik*, 131(2):329–368, 2015.

[22] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

[23] Stefan Heinrich. Multilevel Monte Carlo methods. *Large-scale scientific computing*, pages 58–67, 2001.

[24] Ralf Hiptmair and Lars Kielhorn. Betl — a generic boundary element template library. Technical Report 2012-36, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2012.

[25] Viet Ha Hoang, Christoph Schwab, and Andrew M. Stuart. Complexity analysis of accelerated MCMC methods for Bayesian inversion. *Inverse Problems*, 29(8):085010, 37, 2013.

[26] Torsten Hoefler and Roberto Belli. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 73:1–73:12, New York, NY, USA, 2015. ACM.

[27] Kitware. Cmake. `http://www.cmake.org/`, 2015.

[28] Christiane Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer Series in Statistics. Springer, Dordrecht, 2009.

[29] Gunther Leobacher and Friedrich Pillichshammer. *Introduction to quasi-Monte Carlo integration and applications*. Birkhäuser Basel, 2014.

[30] James A. Nichols and Frances Y. Kuo. Fast CBC construction of randomly shifted lattice rules achieving $\mathcal{O}(n^{-1+\delta})$ convergence for unbounded integrands over $\mathbb{R}^s$ in weighted spaces with POD weights. *Journal of Complexity*, 30(4):444–468, 2014.

[31] Harald Niederreiter. *Random Number Generation and*

Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[32] Dirk Nuyens and Ronald Cools. Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. *Mathematics of Computation*, 75(254):903–920, 2006.

[33] Dirk Nuyens and Ronald Cools. Fast component-by-component construction, a reprise for different kernels. In *Monte Carlo and quasi-Monte Carlo methods 2004*, pages 373–387. Springer, Berlin, 2006.

[34] Dirk Nuyens and Frances Y. Kuo. Application of quasi-Monte Carlo methods to PDEs with random coefficients – analysis and implementation. *(submitted)*, 2015.

[35] Stefan Pauli, Peter Arbenz, and Christoph Schwab. Intrinsic fault tolerance of multilevel Monte Carlo methods. *Journal of Parallel and Distributed Computing*, 84:24 – 36, 2015.

[36] Stefan Pauli, Robert N. Gantner, Peter Arbenz, and Andreas Adelmann. Multilevel Monte Carlo for the Feynman–Kac formula for the Laplace equation. *BIT Numerical Mathematics*, 55(4):1125–1143, 2015.

[37] Claudia Schillings and Christoph Schwab. Sparsity in Bayesian inversion of parametric operator equations. *Inverse Problems*, 30(6):065007, 30, 2014.

[38] Christoph Schwab and Claude Jeffrey Gittelson. Sparse tensor discretizations of high-dimensional parametric and stochastic PDEs. *Acta Numerica*, 20:291–467, 5 2011.

[39] Christoph Schwab and Radu Alexandru Todor. Karhunen-Loève Approximation of Random Fields by Generalized Fast Multipole Methods. *Journal of Computational Physics*, 217(1):100–122, September 2006.

[40] Victor Shoup. *NTL: A Library for doing Number Theory*. Courant Institute, New York University, New York, NY, 2005. Available at `http://shoup.net/ntl/`.

[41] Andrew M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 5 2010.

[42] Jonas Šukys. *Parallel Processing and Applied Mathematics: 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I*, chapter Adaptive Load Balancing for Massively Parallel Multi-Level Monte Carlo Solvers, pages 47–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[43] Jonas Šukys, Siddhartha Mishra, and Christoph Schwab. Static load balancing for multi-level Monte Carlo finite volume solvers. *Parallel Processing and Applied Mathematics*, 0(1):245–254, 2012.