MULTILEVEL MONTE CARLO SAMPLING ON HETEROGENEOUS COMPUTER ARCHITECTURES

Christiane Adcock, [*] ***Yinyu Ye, Lluís Jofre, & Gianluca Iaccarino***

Stanford University, Stanford, California, 94305

*Address all correspondence to: Christiane Adcock, Stanford University, Stanford, California, 94305, E-mail: janiad@stanford.edu

Original Manuscript Submitted: 12/15/2020; Final Draft Received: 9/6/2020

Monte Carlo (MC) sampling is the standard approach for uncertainty propagation in problems with high-dimensional stochastic inputs. Various acceleration techniques have been developed to overcome the slow convergence of MC estimates, such as multilevel Monte Carlo (MLMC). MLMC uses successive approximations computed on levels, models with different levels of accuracy, and computational cost to reduce the estimator variance. MLMC analytically determines the number of samples required on each level to achieve a given accuracy at minimal cost. We propose an extension of the original MLMC theoretical framework for modern, heterogeneous computer architectures in which accelerators (GPUs) are available and, therefore, samples can be distributed on both different levels and different compute units (CPUs and GPUs). We derive the optimal sample allocation for the proposed MLMC extension by solving a convex optimization problem. We apply the MLMC extension to a stochastically heated channel flow to provide insight for a study on the design of concentrated solar energy receivers. We demonstrate for the stochastically heated channel flow that the proposed MLMC extension leads to considerable total cost reduction (up to 86%) compared to MLMC using only GPUs.

KEY WORDS: *uncertainty quantification, variance reduction methods, Monte Carlo, stochastic partial differential equations, stochastic ordinary differential equations, computational design, computational statistics*

1. INTRODUCTION

Uncertainty propagation methods, such as Monte Carlo (MC), aim at computing the outputs of a mathematical model with stochastic inputs [1] by combining model solutions (samples) corresponding to random realizations of the inputs. The convergence rate of Monte Carlo is independent of the dimension of the stochastic input, making it the only computationally tractable method for problems with high-dimensional stochastic inputs. Traditional MC converges slowly—with the square root of the number of samples and describes the output only by an unbiased estimator for the mean. In recent years, research efforts have developed MC-based methods that converge faster and provide flexibility in the description of the output statistics. Multilevel Monte Carlo (MLMC) [2], multifidelity Monte Carlo (MFMC) [3], and control variate Monte Carlo [4] combine solutions from multiple models to reduce the variance and thereby accelerate the convergence of the output estimates. Geraci et al. [5] and Qian et al. [6] extend MLMC and MFMC to estimate not just the mean, but also the variance of the output, and Krumscheid et al. [7] introduces a strategy to estimate moments of any order. Giles et al. [8] and Krumscheid and Nobile [9] propose variations of MLMC to estimate the full distribution of the output.

While much progress has been made on improving the convergence and descriptiveness of MC-type methods, almost no effort has been devoted to taking advantage of emerging computer architectures. Historically, each sample of a MC-type simulation was run on a single CPU. Modern computer architectures, including seven of the top ten supercomputers in the world [10], use heterogeneous nodes, with multiple CPUs and one or more GPUs. In recent work, [11,12] used GPUs in studies with MC and a variant, quasi-Monte Carlo, and [13] used a truly heterogeneous system for MC. Given the typical acceleration afforded by GPUs over CPUs, most current approaches compute samples solely on GPUs. If the software supports executions on CPUs and GPUs, it is possible to instead compute samples on both processor types, thereby taking advantage of all available compute resources. However, there is no theoretical framework on heterogeneous computer architectures, either for MC or for MLMC, to determine the sample allocation that meets a given accuracy at minimal computational cost. We propose a generalization of MLMC, named H-MLMC, which provides this framework. We describe the optimization problem to determine the sample allocation for H-MLMC in Section 2, outline the solution in Section 3, and fully derive the solution in the Appendix. Section 4 demonstrates the computational savings and properties of H-MLMC. Section 5 provides concluding remarks.

2. PROBLEM DEFINITION

2.1 Optimization Problem

We are interested in finding an unbiased estimator of the solution to a stochastic partial differential equation (SPDE). We want the variance of the estimator to be smaller than the tolerance ε^2, and the overall computational cost (total wall-clock time) of the estimator to be as small as possible. We have L numerical models available to solve the SPDE. The models have decreasing variance and increasing computational cost. We assume that each model can be solved on two different types of computer processors, such as CPUs and GPUs. We denote the processor types as C and G processors, respectively. Let N_ℓ^C and N_ℓ^G be the number of samples we solve on level ℓ on C and G processors, respectively. We also assume that the overall compute system has M^C and M^G processors of C and G types, respectively.

An effective method to find an unbiased estimator of the solution to SPDEs is multilevel Monte Carlo (MLMC) [2]. However, MLMC does not take into account the availability of multiple processors of different types when determining the optimal sample allocation. MLMC assumes all samples on a level will be evaluated on the same processor type and finds the optimal number of samples per level under this assumption. A simplistic approach to use MLMC on a heterogeneous computer architecture is to assume that G processors are consistently faster at solving the SPDE than C processors, and to therefore use only G processors. Under this assumption the MLMC estimator [2] is

$$\hat{Q}^{\text{MLMC}} = \frac{1}{N_1^G}\sum_{n=1}^{N_1} Q_1^{G,(n)} + \sum_{\ell=2}^{L}\left(\frac{1}{N_\ell^G}\sum_{n=1}^{N_\ell^G}\left(Q_\ell^{G,(n)} - Q_{\ell-1}^{G,(n))}\right)\right), \tag{1}$$

where $Q_1^{G,(n)}$ is the solution to the SPDE found by model 1 on a G processor for the nth of N_1 independent samples of the stochastic parameters. $Q_\ell^{(n)} - Q_{\ell-1}^{(n)}$ is the difference between the solutions to the SPDE found by models ℓ and $\ell-1$ on a G processor both for the same realization of the stochastic parameters. N_ℓ^G is the number of samples solved on level ℓ on G processors.

The total variance, cost, and wall-clock time to solution of the MLMC estimator are

$$V[\hat{Q}^{\text{MLMC}}] = \sum_{\ell=1}^{L}\frac{V_\ell}{N_\ell^G}, \tag{2}$$

$$C[\hat{Q}^{\text{MLMC}}] = \sum_{\ell=1}^{L} C_\ell^G N_\ell^G, \tag{3}$$

$$T^{\text{MLMC}} = \frac{1}{M^G}\sum_{\ell=1}^{L} C_\ell^G N_\ell^G, \tag{4}$$

respectively. $V_1 = V[Q_1^C] = V[Q_1^G]$ denotes the variance of a sample on level 1 and $V_\ell = V[Q_\ell^C - Q_{\ell-1}^C] = V[Q_\ell^G - Q_{\ell-1}^G]\ \forall l > 1$ denotes the variance of the difference between a sample on levels ℓ and $\ell-1$. $C_1^G = C^G[Q_1^G]$ denotes the cost for a sample on level 1 and $C_\ell^G = C^G[Q_\ell^G - Q_{\ell-1}^G]$ denotes the cost for a sample on levels ℓ and $\ell-1$, each evaluated with a G processor.

We now derive a theoretical framework to extend MLMC by explicitly taking into account the availability of heterogeneous processors. In this new approach, denoted H-MLMC, the estimator is

$$\begin{aligned}\hat{Q}^{\text{H-MLMC}} &= \frac{1}{N_1^C + N_1^G}\left(\sum_{n=1}^{N_1^C} Q_1^{C,(n)} + \sum_{n=1}^{N_1^G} Q_1^{G,(n)}\right)\\ &+ \sum_{\ell=2}^{L}\left(\frac{1}{N_\ell^C + N_\ell^G}\left(\sum_{n=1}^{N_\ell^C}\Big(Q_\ell^{C,(n)} - Q_{\ell-1}^{C,(n))}\Big) + \sum_{n=1}^{N_\ell^G}\Big(Q_\ell^{G,(n)} - Q_{\ell-1}^{G,(n))}\Big)\right)\right).\end{aligned} \tag{5}$$

When generating realizations of the stochastic parameter, care should be taken so that the same stochastic parameter is not used twice—this was done for all examples presented in this paper. The total variance, cost, and wall-clock time to solution of the H-MLMC estimator are

$$V[\hat{Q}^{\text{H-MLMC}}] = \sum_{\ell=1}^{L} \frac{V_\ell}{N_\ell^C + N_\ell^G}, \tag{6}$$

$$C[\hat{Q}^{\text{H-MLMC}}] = \sum_{\ell=1}^{L}\big(C_\ell^C N_\ell^C + C_\ell^G N_\ell^G\big), \tag{7}$$

$$T^{\text{H-MLMC}} = \max\left(\frac{1}{M^C}\sum_{\ell=1}^{L} C_\ell^C N_\ell^C, \frac{1}{M^G}\sum_{\ell=1}^{L} C_\ell^G N_\ell^G\right). \tag{8}$$

where C_ℓ^C is defined analogously to C_ℓ^G, with samples evaluated on C not G processors. We seek the optimal sample allocation: the values of N_ℓ^C and N_ℓ^G $\forall \ell = 1, 2, ..., L$ that minimize the wall-clock time such that the total variance is less than ε^2. N_ℓ^C and N_ℓ^G must be integers. However, integer programming is NP-complete. Therefore, we remove the constraint that N_ℓ^C and N_ℓ^G are integers, as done when deriving the optimal sample allocation for MLMC [2].

The optimization problem is

$$\begin{aligned}&\min_{N_1^C, N_2^C, \ldots, N_L^C, N_1^G, N_2^G, \ldots, N_L^G} \max\left(\frac{1}{M^C}\sum_{\ell=1}^{L} N_\ell^C C_\ell^C, \frac{1}{M^G}\sum_{\ell=1}^{L} N_\ell^G C_\ell^G\right),\\ &\text{s.t.} \sum_{\ell=1}^{L} \frac{V_\ell}{N_\ell^C + N_\ell^G} \le \varepsilon^2,\\ &\quad 0 \le N_\ell^C \;\forall \ell = 1, 2, ..., L,\\ &\quad 0 \le N_\ell^G \;\forall \ell = 1, 2, ..., L.\end{aligned} \tag{9}$$

2.2 Proof of Convexity

Equation (9) can be rewritten as

$$\min_{N_1^C, N_2^C, \ldots, N_L^C, N_1^G, N_2^G, \ldots, N_L^G, T} T, \tag{10}$$

$$\text{s.t.} -\sum_{\ell=1}^{L}\left(\frac{V_\ell}{N_\ell^C + N_\ell^G}\right) + \varepsilon^2 \ge 0, \tag{11}$$

$$-\frac{1}{M^C}\sum_{\ell=1}^{L}\big(C_\ell^C N_\ell^C\big) + T \ge 0, \tag{12}$$

$$-\frac{1}{M^G}\sum_{\ell=1}^{L}\big(C_\ell^G N_\ell^G\big) + T \ge 0, \tag{13}$$

$$N_\ell^C \ge 0 \;\forall \ell = 1, 2, ..., L, \tag{14}$$

$$N_\ell^G \ge 0 \;\forall \ell = 1, 2, ..., L. \tag{15}$$

Equations (10)–(15) define a convex optimization problem. Equations (10) and (12)–(15) are linear, so they are convex. Equation (11) is equivalent to:

$$\sum_{\ell=1}^{L}\left(\frac{V_\ell}{N_\ell^C + N_\ell^G}\right) - \varepsilon^2 \leq 0. \tag{16}$$

Equation (16) has the Hessian:

$$H = \nabla^2_{(N_1^C, N_2^C, \ldots, N_L^C, N_1^G, N_2^G, \ldots, N_L^G)}\left(\sum_{\ell=1}^{L}\left(\frac{V_\ell}{N_\ell^C + N_\ell^G}\right) - \varepsilon^2\right)$$

$$= \begin{bmatrix} \frac{2V_1}{(N_1^C+N_1^G)^3} & & & \frac{2V_1}{(N_1^C+N_1^G)^3} & & \\ & \frac{2V_2}{(N_2^C+N_2^G)^3} & & & \frac{2V_2}{(N_2^C+N_2^G)^3} & \\ & & \ddots & & & \ddots \\ & & \frac{2V_L}{(N_L^C+N_L^G)^3} & & & \frac{2V_L}{(N_L^C+N_L^G)^3} \\ \frac{2V_1}{(N_1^C+N_1^G)^3} & & & \frac{2V_1}{(N_1^C+N_1^G)^3} & & \\ & \frac{2V_2}{(N_2^C+N_2^G)^3} & & & \frac{2V_2}{(N_2^C+N_2^G)^3} & \\ & & \ddots & & & \ddots \\ & & \frac{2V_L}{(N_L^C+N_L^G)^3} & & & \frac{2V_L}{(N_L^C+N_L^G)^3} \end{bmatrix}.$$

Assuming that all $V_\ell \neq 0$, H is real and symmetric, so it has real eigenvalues. By the Gershgorin circle theorem, for each eigenvalue of H, $\lambda(H) \in \left[H_{ii} - \sum_{j\neq i} H_{ij}, H_{ii} + \sum_{j\neq i} H_{ij}\right]$ for some i. By inspection H is diagonally dominant and has non-negative entries, so $H_{ii} - \sum_{j\neq i} H_{ij} \geq 0$. Combining results from the Gershgorin circle theorem and diagonal dominance, $\lambda(H) \geq 0$. $det(H) \neq 0$ so $\lambda(H) \neq 0$. Therefore, $\lambda(H) > 0$, meaning H is symmetric positive-definite. Thus Eq. (16), and equivalently Eq. (11), are convex.

2.3 Karush, Kuhn, Tucker (KKT) Conditions

The optimization problem is convex, so the KKT first-order necessary conditions are sufficient for optimality. The KKT conditions are

$$\begin{aligned} \nabla\Bigg(T - \lambda_1\left(-\sum_{\ell=1}^{L}\left(\frac{V_\ell}{N_\ell^C + N_\ell^G}\right) + \varepsilon^2\right) - \lambda_2\left(-\frac{1}{M^C}\sum_{\ell=1}^{L}(C_\ell^C N_\ell^C) + T\right) \\ - \lambda_3\left(-\frac{1}{M^G}\sum_{\ell=1}^{L}(C_\ell^G N_\ell^G) + T\right) - \sum_{\ell=1}^{L}(\mu_\ell^C N_\ell^C) - \sum_{\ell=1}^{L}(\mu_\ell^G N_\ell^G)\Bigg) = 0, \end{aligned} \tag{17}$$

$$\lambda_1\left(-\sum_{\ell=1}^{L}\left(\frac{V_\ell}{N_\ell^C + N_\ell^G}\right) + \varepsilon^2\right) = 0, \tag{18}$$

$$\lambda_2\left(-\frac{1}{M^C}\sum_{\ell=1}^{L}(C_\ell^C N_\ell^C) + T\right) = 0, \tag{19}$$

$$\lambda_3\left(-\frac{1}{M^G}\sum_{\ell=1}^{L}(C_\ell^G N_\ell^G)+T\right)=0, \tag{20}$$

$$\mu_\ell^C N_\ell^C = 0\ \forall \ell = 1,2,\ldots,L, \tag{21}$$

$$\mu_\ell^G N_\ell^G = 0\ \forall \ell = 1,2,\ldots,L, \tag{22}$$

$$\lambda_i \geq 0\ \forall i = 1,2,3, \tag{23}$$

$$\mu_\ell^C \geq 0\ \forall \ell = 1,2,\ldots,L, \tag{24}$$

$$\mu_\ell^G \geq 0\ \forall \ell = 1,2,\ldots,L, \tag{25}$$

$$N_\ell^C \geq 0\ \forall \ell = 1,2,\ldots,L, \tag{26}$$

$$N_\ell^G \geq 0\ \forall \ell = 1,2,\ldots,L. \tag{27}$$

The problem is now to find the optimal sample allocation, N_ℓ^C and $N_\ell^G\ \forall \ell = 1,2,\ldots,L$, and Lagrange multipliers, λ_i for $i = 1,2,3$, $\mu_\ell^C\ \forall \ell = 1,2,\ldots,L$, and $\mu_\ell^G\ \forall \ell = 1,2,\ldots,L$, that satisfy Eqs. (17)–(27).

3. OPTIMAL SAMPLE ALLOCATION

3.1 Monte Carlo on Heterogeneous Compute Nodes (H-MC)

Consider the simplest case of the optimization problem in Section 2, where there is one level, $L = 1$. This is equivalent to finding the optimal sample allocation for Monte Carlo when heterogeneous computing resources are available.

Theorem 1. *The optimal H-MC sample allocation, which solves Eq. (9) for $L = 1$ is*

$$N_1^C = \frac{V_1 C_1^G M^C}{\varepsilon^2(C_1^C M^G + C_1^G M^C)};\quad N_1^G = \frac{V_1 C_1^C M^G}{\varepsilon^2(C_1^C M^G + C_1^G M^C)}.$$

Proof. Equations (17)–(27) constrain $\lambda_1, \lambda_2 > 0$, and $\lambda_3 > 0$, as proved in the Appendix for the general case where $L \geq 1$. Since λ_2 and $\lambda_3 > 0$, by Eqs. (19) and (20),

$$\frac{C_1^C N_1^C}{M^C} = \frac{C_1^G N_1^G}{M^G}. \tag{28}$$

Since $\lambda_1 > 0$, by Eq. (18),

$$\frac{V_1}{N_1^C + N_1^G} = \varepsilon^2. \tag{29}$$

Solving Eqs. (28) and (29) for N_1^C and N_1^G completes the proof. $\square$

Equation (28) leads to the observation that although G processors are faster than C processors, it is worth using both rather than just the former as done in [11,12]. Ensuring that computations on the C and G processors complete at the same time reduces the overall cost of the estimator.

3.2 Multilevel Monte Carlo on Heterogeneous Compute Nodes (H-MLMC)

Consider the optimization problem in Section 2, initially for two levels, $L = 2$.

By Eqs. (19) and (20),

$$\frac{C_1^C N_1^C + C_2^C N_2^C}{M^C} = \frac{C_1^G N_1^G + C_2^G N_2^G}{M^G}. \tag{30}$$

By Eq. (18),

$$\frac{V_1}{N_1^C + N_1^G} + \frac{V_2}{N_2^C + N_2^G} = \varepsilon^2. \tag{31}$$

Unlike the one-level case described in Section 3.1, we now have four unknowns: N_1^C, N_1^G, N_2^C, and N_2^G. Therefore, we cannot uniquely determine the optimal sample allocation just from these two equations. Instead, we must consider additional constraints derived from the KKT conditions. The same reasoning holds for $L > 2$. Using $L = 2$ does not simplify the equations significantly, so for the remainder of this section we present the general results for $L \geq 1$.

Theorem 2. *Any sample allocation N_ℓ^C and N_ℓ^G $\forall \ell = 1, 2, ..., L$ which satisfies Eqs. (32)–(38), solves Eq. (9), and is therefore an optimal sample allocation for H-MLMC.*

$$\frac{1}{M^C}\sum_{\ell=1}^{L}(C_\ell^C N_\ell^C) = \frac{1}{M^G}\sum_{\ell=1}^{L}(C_\ell^G N_\ell^G), \tag{32}$$

$$\sum_{\ell=1}^{L}\frac{V_\ell}{N_\ell^C + N_\ell^G} = \varepsilon^2, \tag{33}$$

$$\lambda_1 = \left(\frac{1}{\varepsilon^2}\sum_{\ell=1}^{L}\sqrt{\frac{V_\ell}{\max\big(1/(\lambda_2 C_\ell^C M^G), 1/(\lambda_3 C_\ell^G M^C)\big)}}\right)^2, \tag{34}$$

$$1 = \lambda_2 + \lambda_3, \tag{35}$$

$\forall \ell = 1, 2, ..., L$:

$$\text{CASE 1: if } \frac{\lambda_3}{\lambda_2} > \frac{C_\ell^C M^G}{C_\ell^G M^C} \text{ then } \begin{cases} N_\ell^C = \sqrt{\dfrac{\lambda_1 V_\ell}{\lambda_2 C_\ell^G M^C}} \\ N_\ell^G = 0 \end{cases}, \tag{36}$$

$$\text{CASE 2: if } \frac{\lambda_3}{\lambda_2} < \frac{C_\ell^C M^G}{C_\ell^G M^C} \text{ then } \begin{cases} N_\ell^C = 0 \\ N_\ell^G = \sqrt{\dfrac{\lambda_1 V_\ell}{\lambda_3 C_\ell^C M^G}} \end{cases}, \tag{37}$$

$$\text{CASE 3: if } \frac{\lambda_3}{\lambda_2} = \frac{C_\ell^C M^G}{C_\ell^G M^C} \text{ then } \begin{cases} N_\ell^C \in \left[0, \sqrt{\dfrac{\lambda_1 V_\ell}{\lambda_2 C_\ell^G M^C}}\right] \\ N_\ell^G = \sqrt{\dfrac{\lambda_1 V_\ell}{\lambda_2 C_\ell^G M^C}} - N_\ell^C \end{cases}. \tag{38}$$

Proof. See the Appendix for the proof. □

λ_2 and λ_3 cannot be found analytically from Theorem 2. Therefore, we introduce an iterative procedure to find λ_2 and λ_3, and consequently λ_1 and N_ℓ^C and N_ℓ^G $\forall \ell = 1, 2, ..., L$. To construct the procedure, we observe the impact of λ_3/λ_2 on Eq. (32). If this ratio is sufficiently small $\big[$less than $\alpha_L = \min_{\ell=1,2...,L}(C_\ell^C M^G / C_\ell^G M^C)\big]$, then all levels will fall into case 2. All samples will be solved on the G processor, so the right-hand side of Eq. (32) will be larger than the left-hand side (which will equal zero).

For sufficiently large λ_3/λ_2, some level l^* will fall into case 3. Some samples on level ℓ^* will be solved on the C processor and some on the G processor. As the ratio further increases, more ℓ^* samples will be solved on the C processor instead of the G processor. For sufficiently large ratios, ℓ^* will fall into case 1, so all ℓ^* samples will be solved on the C processor. As the ratio increases more levels will switch from case 2 to case 3 and then to case 1. Each of these changes increases the left-hand side and decreases the right-hand side of Eq. (32).

Finally, if λ_3/λ_2 is sufficiently large [greater than $\alpha_R = \max_{\ell=1,2,\ldots,L}(C_\ell^C M^G/C_\ell^G M^C)$] then all levels will fall into case 1. All the samples will be solved on the C processor, so the left-hand side of Eq. (32) will be larger than the right-hand side (which will equal zero).

Equation (32) can only be satisfied for the more complex intermediate case of $\lambda_3/\lambda_2 \in [\alpha_L, \alpha_R]$, not for the extreme cases of $\lambda_3/\lambda_2 = \alpha_L$ and $\lambda_3/\lambda_2 = \alpha_R$. To find the optimal sample allocation we apply a bisection strategy as described in Algorithm 1. Step 5 of Algorithm 1 requires additional reasoning, which is provided in the Appendix.

From Theorem 2, we determine under which conditions N_ℓ^C and N_ℓ^G are not unique on any level and under which conditions they could be unique for all levels. We report these conditions in Corollaries 1 and 2, respectively.

Corollary 1. *C_ℓ^C/C_ℓ^G is the same for all levels iff N_ℓ^C and N_ℓ^G are not unique on any level*

Proof. If C_ℓ^C/C_ℓ^G is the same for all levels, then

$$\frac{\lambda_3}{\lambda_2} \in [\alpha_L, \alpha_R] = \left[\frac{C_\ell^C M^G}{C_\ell^G M^C}, \frac{C_\ell^C M^G}{C_\ell^G M^C}\right], \text{ so } \frac{\lambda_3}{\lambda_2} = \frac{C_\ell^C M^G}{C_\ell^G M^C} \; \forall l.$$

Therefore, by Eq. (38), N_ℓ^C and N_ℓ^G are not unique on any level.

If N_ℓ^C and N_ℓ^G are not unique on any level, then by Eqs. (36) and (37), $(\lambda_3/\lambda_2)(\not>, \not<)(C_\ell^C M^G/C_\ell^G M^C)$. Therefore, $C_\ell^C M^G/C_\ell^G M^C = \lambda_3/\lambda_2$, so C_ℓ^C/C_ℓ^G is the same for all levels. □

Corollary 2. *If N_ℓ^C and N_ℓ^G are uniquely defined for all levels then C_ℓ^C/C_ℓ^G cannot be the same for all levels.*

Proof. By the contrapositive of the if statement of Corollary 1, if N_ℓ^C and N_ℓ^G are uniquely defined for some level then C_ℓ^C/C_ℓ^G cannot be the same for all levels. If N_ℓ^C and N_ℓ^G are uniquely defined for all levels then N_ℓ^C and N_ℓ^G are uniquely defined for some level, so C_ℓ^C/C_ℓ^G cannot be the same for all levels. □

Algorithm 1: H-MLMC Sample Allocation

```
$\alpha = \left(\frac{C_1^C}{C_1^G} + \frac{C_L^C}{C_L^G}\right)/2$
While $\frac{1}{M^C}\sum_{\ell=1}^{L} C_\ell^C N_\ell^C \neq \frac{1}{M^G}\sum_{\ell=1}^{L} C_\ell^G N_\ell^G$:
    Step 1. Set $\frac{\lambda_3}{\lambda_2} = \alpha$ by bisection method
    Step 2. $\lambda_2 = \frac{1}{1+\alpha}$ and $\lambda_3 = \frac{\alpha}{1+\alpha}$
    Step 3. $\lambda_1 = \left(\frac{1}{\varepsilon^2}\sum_{\ell=1}^{L}\sqrt{\frac{V_\ell}{\max\left(1/(\lambda_2 C_\ell^C M^G), 1/(\lambda_3 C_\ell^G M^C)\right)}}\right)^2$
    Step 4. Set $N_\ell^C$ and $N_\ell^G$ $\forall \ell \in S$, where $S = \left\{ l \middle| \frac{C_\ell^C M^G}{C_\ell^G M^C} \neq \frac{\lambda_3}{\lambda_2} \right\}$:
        If $\frac{\lambda_3}{\lambda_2} > \frac{C_\ell^C M^G}{C_\ell^G M^C}$ then $\begin{cases} N_\ell^C = \sqrt{\frac{\lambda_1 V_\ell}{\lambda_2 C_\ell^G M^C}} \\ N_\ell^G = 0 \end{cases}$
        If $\frac{\lambda_3}{\lambda_2} < \frac{C_\ell^C M^G}{C_\ell^G M^C}$ then $\begin{cases} N_\ell^C = 0 \\ N_\ell^G = \sqrt{\frac{\lambda_1 V_\ell}{\lambda_3 C_\ell^C M^G}} \end{cases}$
    Step 5. Set $N_\ell^C$ and $N_\ell^G$ $\forall l \notin S$ to minimize $\frac{1}{M^C}\sum_{\ell \in S} C_\ell^C N_\ell^C - \frac{1}{M^G}\sum_{\ell \in S} C_\ell^G N_\ell^G$.
```

Using Corollary 1, we can determine *a priori* when N_ℓ^C and N_ℓ^G are not unique on any level. We do not have a way to determine *a priori* when N_ℓ^C and N_ℓ^G are unique on all levels. All we know is that Corollary 2 must be satisfied for our optimal sample allocation to be unique.

From Theorem 2, we also find that when C_ℓ^C/C_ℓ^G is the same for all levels, the total number of samples on each level equals the optimal sample allocation for MLMC: $N_\ell^C + N_\ell^G = (N_\ell^G)^{\text{MLMC}}$. N_ℓ^C and N_ℓ^G are the optimal sample allocation from H-MLMC and $(N_\ell^G)^{\text{MLMC}}$ is the optimal sample allocation from MLMC. Corollary 3 formally states and proves this finding.

Corollary 3. *If $C_\ell^C/C_\ell^G = \alpha\ \forall \ell = 1, 2, ..., L$ for some constant α, $N_\ell^C + N_\ell^G$ from H-MLMC equals N_ℓ^G from MLMC, $(N_\ell^G)^{MLMC}$.*

Proof. By Eq. (38),

$$N_\ell^C + N_\ell^G = \sqrt{\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}}.$$

By Eqs. (34) and (A.28) from the Appendix,

$$\frac{\lambda_1}{\lambda_2} = \left(\frac{1}{\varepsilon^2}\sum_{i=1}^{L}\sqrt{V_i\frac{1}{1+\alpha}C_i^C M^G}\right)^2(1+\alpha).$$

Combining results and using that $C_\ell^C = \alpha C_\ell^G$,

$$\begin{aligned} N_\ell^C + N_\ell^G &= \frac{1}{\varepsilon^2}\sum_{i=1}^{L}\sqrt{V_i\frac{1}{1+\alpha}\alpha C_i^G M^G}\sqrt{1+\alpha}\sqrt{\frac{V_\ell}{\alpha C_\ell^G M^G}} \\ &= \sqrt{\frac{V_\ell}{C_\ell^G}}\frac{1}{\varepsilon^2}\sum_{i=1}^{L}\sqrt{V_i C_i^G} = (N_\ell^G)^{\text{MLMC}}. \end{aligned}$$

□

From Corollary 3, we find that the ratio of the wall-clock time for H-MLMC, $T^{\text{H-MLMC}}$, to that for MLMC, T^{MLMC}, depends only on the ratio C_ℓ^C/C_ℓ^G, not on the values of C_ℓ^C, C_ℓ^G, or $V_\ell\ \forall \ell = 1, 2, ..., L$ or on L. In addition, $T^{\text{H-MLMC}}/T^{\text{MLMC}} < 1$. Corollary 4 states and proves these findings.

Corollary 4. *If $C_\ell^C/C_\ell^G = \alpha\ \forall \ell = 1, 2, ..., L$ for some constant α, then $T^{H-MLMC}/T^{MLMC} = 1/[1 + (1/\alpha)(M^C/M^G)] < 1$*

Proof.

$$\frac{T^{\text{H-MLMC}}}{T^{\text{MLMC}}} = \frac{(1/M^G)\sum_{\ell=1}^{L} C_\ell^G N_\ell^G}{(1/M^G)\sum_{\ell=1}^{L} C_\ell^G (N_\ell^G)^{\text{MLMC}}}.$$

By Corollary 3, this equals

$$\frac{\sum_{\ell=1}^{L} C_\ell^G N_\ell^G}{\sum_{\ell=1}^{L} C_\ell^G (N_\ell^C + N_\ell^G)}.$$

By Eqs. (A.36) and (A.37) from the Appendix, this equals

$$\sum_{\ell=1}^{L} C_\ell^G N_\ell \frac{C_\ell^C M^G}{C_\ell^C M^G + C_\ell^G M^C}\Big/\sum_{\ell=1}^{L} C_\ell^G N_\ell = \sum_{\ell=1}^{L} C_\ell^G N_\ell\left[1\Big/\left(1+\frac{1}{\alpha}\frac{M^C}{M^G}\right)\right]\Big/\sum_{\ell=1}^{L} C_\ell^G N_\ell = 1\Big/\left(1+\frac{1}{\alpha}\frac{M^C}{M^G}\right) < 1.$$

where the last inequality holds because α, M^C, and M^G are nonnegative. □

3.3 Distributing Samples on Multiple Processors

Sections 2, 3.1, and 3.2 derive the optimal number of samples to solve on each level with C processors and with G processors. However, they do not specify how many processors to use to solve each sample. We've defined M^C and M^G to be the total number of C processors and G processors. Let m_ℓ^C and m_ℓ^G be the number of processors used to solve each C processor and G processor sample on level ℓ, respectively. There are four approaches to assigning m_ℓ^C and m_ℓ^G:

1. Solve every sample with one processor: $m_\ell^C = m_\ell^G = 1 \; \forall \ell = 1, 2, ..., L$.
2. Solve every sample with all available processors: $m_\ell^C = M^C$ and $m_\ell^G = M^G \; \forall \ell = 1, 2, ..., L$, and assume perfect scalability.
3. Solve every sample with all available processors: $m_\ell^C = M^C$ and $m_\ell^G = M^G \; \forall \ell = 1, 2, ..., L$, and replace C_ℓ^C and C_ℓ^G with $p_\ell^C C_\ell^C$ and $p_\ell^G C_\ell^G$, where $p_\ell^C > 1$ and $p_\ell^G > 1$ are scalar penalties due to loss of scalability.
4. Modify our optimization problem to also calculate m_ℓ^C and m_ℓ^G. This requires estimating functions $p_\ell^C\left(m_\ell^C\right)$ and $p_\ell^G\left(m_\ell^G\right)$.

H-MLMC uses approach 2. By dividing $\sum_{\ell=1}^{L} C_\ell^G N_\ell^G$ by M^G in Eqs. (4) and (8) and by dividing $\sum_{\ell=1}^{L} C_\ell^C N_\ell^C$ by M^C in Eq. (8), we are assuming that one sample uses an arbitrary number of processors with perfect scalability; i.e., the cost of solving a sample with cost C on m processors is C/m, regardless of how large m is.

Approach 3 simply requires replacing constants C_ℓ^C and C_ℓ^G with the constants $p_\ell^C C_\ell^C$ and $p_\ell^G C_\ell^G$. This approach will find the optimal sample allocation for $m_\ell^C = M^C$ and $m_\ell^G = M^G$. However, different values of m_ℓ^C, m_ℓ^G, N_ℓ^C, and N_ℓ^G might give a smaller solution time.

H-MLMC approximates approach 1 when condition (a), subsequently defined, holds. Condition (a): there is enough work to use all the available processors for at least as long as it takes to run a sample on the finest level, and there are orders of magnitude more samples on coarser, low-cost levels than on finer, high-cost levels. For approach 1, this means high-cost samples can be allocated to different processors and then lower-cost samples can be allocated to even out the work on each processor.

Approach 4 is only particularly useful when condition (a) does not hold. In this case our solution time would be reduced by running the samples on the finest level (and potentially then also coarser levels) on more processors, even if this leads to imperfect scalability. Finding the optimal sample allocation for this more general problem would lead to different KKT conditions than those presented in Section 2.3 and a different algorithm for finding the optimal sample allocation than Algorithm 1.

While the perfect scalability assumption in approach 2 is rarely valid in practice, this approach is often still a reasonable approximation. Consider again the scenario when condition (a) holds. Say that in case A we run a sample on level ℓ with so many processors, m_ℓ^A, that it scales imperfectly and takes time T_ℓ^A. If in case B we instead run that sample on $m_\ell^B < m_\ell^A$ processors so that it scales perfectly and takes time T_ℓ^B, then the total processor time will be $m_\ell^B T_\ell^B < m_\ell^A T_\ell^A$. The freed-up time $m_\ell^A T_\ell^A - m_\ell^B T_\ell^B$ available across processors could be filled by some of the many, quick-running coarse samples, reducing the wall-clock time to solution of the H-MLMC estimator, $T^{\text{H-MLMC}}$. Figure 1 illustrates this scenario for $T_\ell^B > T_\ell^A$. It is possible that instead $T_\ell^B (<,=) T_\ell^A$; in this case we would still have $m_\ell^B T_\ell^B < m_\ell^A T_\ell^A$. Therefore, $T^{\text{H-MLMC}}$ will be smallest when each sample is run on a number of processors such that it scales perfectly.

So long as all samples are run on only so many processors that they have perfect scalability, $T^{\text{H-MLMC}}$ does not depend on how many processors are used to run each sample. If we think of each sample as a rectangle with width m_ℓ and height equal to the cost per processor, T_ℓ, as in Fig. 1, then the area of the rectangle is the computational cost of that level, C_ℓ. Therefore, the area of that rectangle does not depend on m_ℓ—if m_ℓ decreases, T_ℓ must increase to keep C_ℓ constant. Thus, the total area divided by the total number of processors, which is $T^{\text{H-MLMC}}$, remains fixed.

The logic in this section assumes that we relax the condition that the number of samples allocated to a level or processor is an integer. As explained in Section 2, this is consistent with MLMC, which relaxes the assumption that the number of samples allocated to a level is an integer.

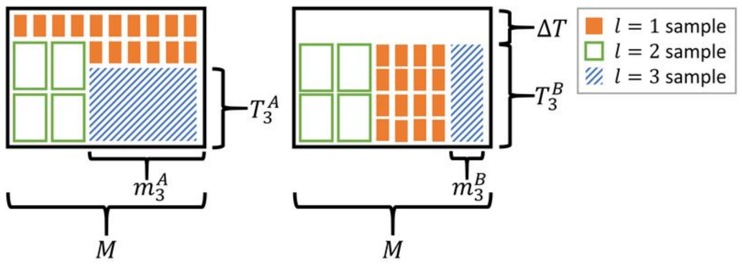

FIG. 1: Each sample is visualized as a rectangle with width equal to the number of processors used to solve that sample, m_ℓ, and height equal to the cost per processor, T_ℓ. Running the sample on the finest level, $l = 3$, on $m_3^B < m_3^A$ rather than m_3^A processors allows the coarse samples, $l = 1$, to fill the freed-up time $m_3^A T_3^A - m_3^B T_3^B$, reducing the wall-clock time to solution of the H-MLMC estimator, $T^{\text{H-MLMC}}$, by ΔT.

4. RESULTS

In the following examples we demonstrate H-MLMC first under the assumption that $C_\ell^C / C_\ell^G = \alpha \; \forall l$, to show the effect of different values of α. Then we demonstrate H-MLMC for a large-scale test case where C_ℓ^C / C_ℓ^G is different on each level: channel flow with a stochastic heat flux boundary condition. The stochastically heated channel flow test case provides insight for a study on the design of concentrated solar energy receivers at the Stanford PSAAP II program. In both examples, C and G processors refer to CPUs and GPUs, respectively.

4.1 Example 1: $C_\ell^C / C_\ell^G = \alpha \forall l$

Example 1 illustrates the often reasonable assumption that the GPU accelerates each level by the same constant factor. We use $\varepsilon^2 = 10^{-2}$ and the GPU cost and variance values listed in Table 1. These tolerance, cost, and variance values are in line with those observed in MLMC studies, such as [14,15]. The CPU cost is $C_\ell^C = \alpha C_\ell^G$ for $\alpha \in [5, 50]$, which spans the range of common GPU to CPU cost ratios. For example, Lassen, the number 10 supercomputer in the world as of November 2019, has a peak GPU TFLOPS to peak CPU TFLOPS ratio of 26 [16]. We use the number of CPUs and GPUs available on Lassen's compute nodes, $M^C = 34{,}848$ and $M^G = 3168$.

The optimal sample allocation and therefore the wall-clock time to solution for H-MLMC, but not for MC or MLMC, depends on C_ℓ^C / C_ℓ^G. In this example $C_\ell^C / C_\ell^G = \alpha \; \forall \ell = 1, 2, ..., L$. Figure 2 shows that MLMC and even more so H-MLMC are faster than MC. For the expected range of GPU to CPU cost ratios, 1-level H-MLMC takes 31–82% as long as MC. Two-level MLMC takes 11% as long as MC, while 2-level H-MLMC takes 3–9% as long as MC. H-MLMC offers more improvement over MLMC when the CPU cost is closer to the GPU cost, because a larger portion of the work in H-MLMC can be done on the CPUs left idle by MLMC.

The allocation of samples between CPUs and GPUs depends on C_ℓ^C / C_ℓ^G. Figure 3 shows this relationship for two-level H-MLMC with the two finest levels of Example 1. As C_ℓ^C / C_ℓ^G increases, fewer samples are solved on the CPU and more samples are solved on the GPU. By Corollary 3, the total number of samples on each level, $N_\ell^C + N_\ell^G = (N_\ell^G)^{\text{MLMC}}$. $(N_\ell^G)^{\text{MLMC}}$ and therefore $N_\ell^C + N_\ell^G$ does not depend on C_ℓ^C / C_ℓ^G.

By Corollary 4, the ratio of the H-MLMC wall-clock time to the MLMC wall-clock time, $T^{\text{H-MLMC}} / T^{\text{MLMC}}$, depends only on C_ℓ^C / C_ℓ^G; Figure 4 illustrates this relationship. As explained for Fig. 2, as C_ℓ^C / C_ℓ^G decreases, H-MLMC offers more improvement over MLMC. For the expected range of GPU to CPU cost ratios, H-MLMC takes 9–82% as long as MLMC.

TABLE 1: Example 1 cost and variance values

l	**1**	**2**	**3**	**4**	**5**
C_ℓ^G	2.0×10^{-2}	3.6×10^{-6}	6.3×10^{-5}	1.1×10^{-3}	2.0×10^{-2}
$V[Q_\ell]$	1.0×10^{-1}	5.6×10^{-2}	3.2×10^{-2}	1.8×10^{-2}	1.0×10^{-2}
$V[Q_\ell - Q_{\ell-1}]$	N/A	5.6×10^{-3}	3.2×10^{-3}	1.8×10^{-3}	1.0×10^{-3}

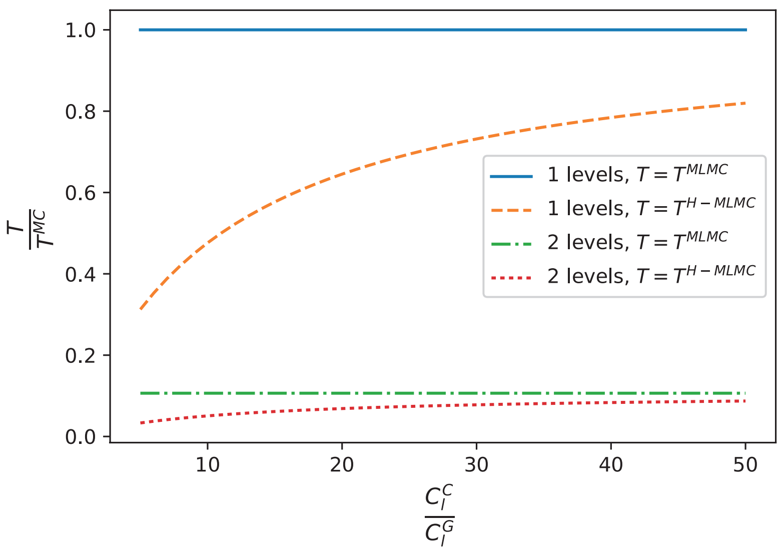

FIG. 2: Ratio of the wall-clock time for MLMC to that for MC, $T^{\text{MLMC}}/T^{\text{MC}}$, and for H-MLMC to that for MC, $T^{\text{H-MLMC}}/T^{\text{MC}}$. Results are for one- and two-level MLMC and H-MLMC for Example 1 using the finest and two finest levels, respectively.

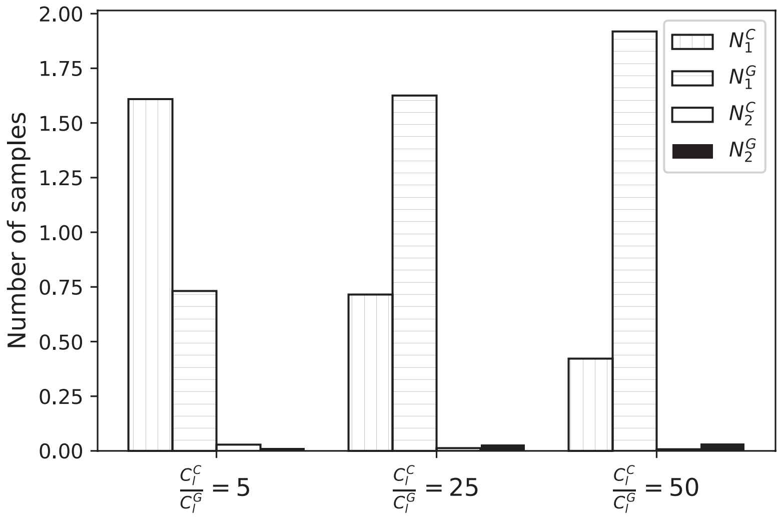

FIG. 3: Optimal sample allocation when $C_\ell^C/C_\ell^G = (5, 25, 50)\ \forall \ell = 1, 2$ for two-level H-MLMC with the two finest levels of Example 1

While $T^{\text{H-MLMC}}/T^{\text{MLMC}}$ does not vary with the number of levels used in MLMC and H-MLMC, T^{MLMC} and $T^{\text{H-MLMC}}$ do. Figures 5 and 6 show that T^{MLMC} and $T^{\text{H-MLMC}}$ decrease as more levels are added, and that the rate of decrease slows rapidly. T^{MLMC} does not depend on C_ℓ^C/C_ℓ^G, while $T^{\text{H-MLMC}}$ does. As C_ℓ^C/C_ℓ^G decreases for fixed C_ℓ^G, $T^{\text{H-MLMC}}$ decreases, since samples can be solved faster on the CPUs.

4.2 Example 2—Channel Flow with Stochastic Heat Flux Boundary Condition: C_ℓ^C/C_ℓ^G Different on Each Level

Example 2 demonstrates H-MLMC when C_ℓ^C/C_ℓ^G is different on each level. This scenario often occurs due to a combination of physics complexity, numerical methods, and memory access. In this example, the GPU speedup

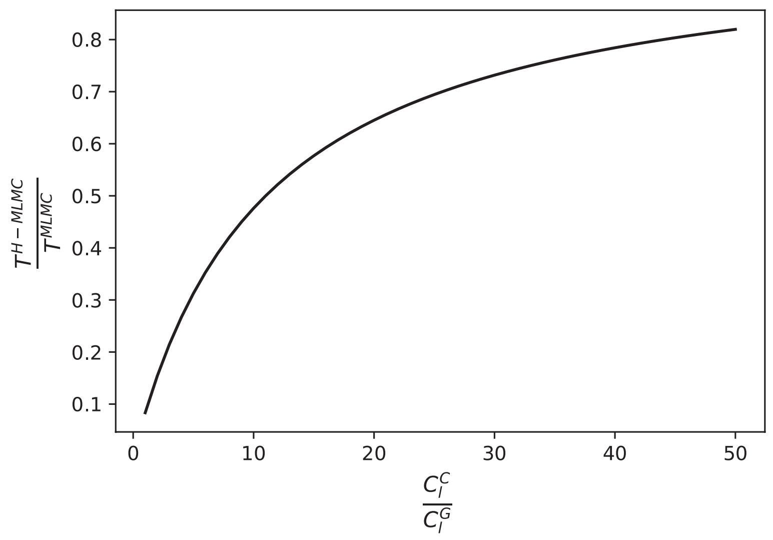

FIG. 4: Ratio of wall-clock time for H-MLMC to that for MLMC, $T^{\text{H-MLMC}}/T^{\text{MLMC}}$, for MLMC and H-MLMC estimators constructed with any number of levels for Example 1

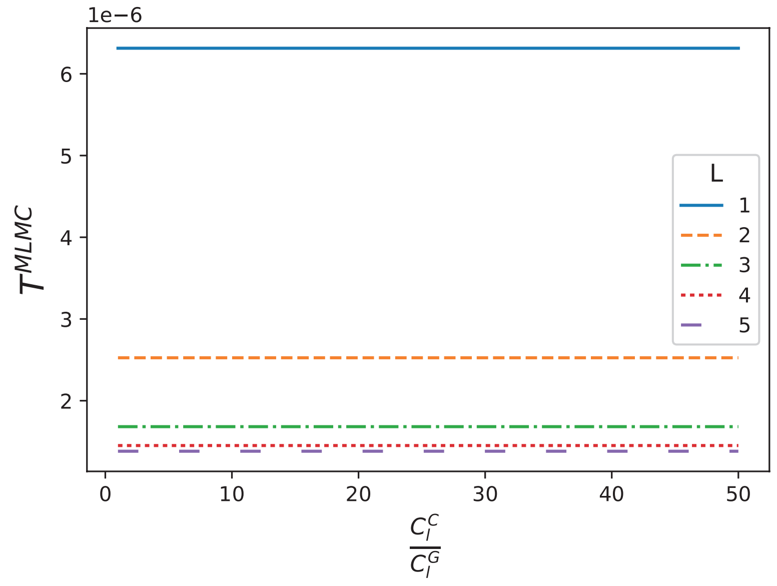

FIG. 5: Wall-clock time for one-, two-, three-, four-, and five-level MLMC for Example 1

increases with grid resolution as the levels become finer. As the levels become finer, the amount of computation relative to communication increases, making the GPUs' computational speed more visible.

This test case provides insight for a study on the design of concentrated solar energy receivers at the Stanford PSAAP II program, described in [17]. The Stanford PSAAP II study developed computational and physical models for particle-laden, irradiated, turbulent flow through a square duct. Physical experiments showed that particles stick to the surface of the duct in random clusters, locally increasing heat transfer. This test case models the effect of particle adhesion in a simplified flow as a first step towards accounting for the effect of particle adhesion.

The test cases uses the three-dimensional multiphysics solver from the PSAAP II study, Soleil-X, with the particles and irradiation turned off and two dimensions: streamwise (x) and one of the spanwise directions (y). Soleil-X

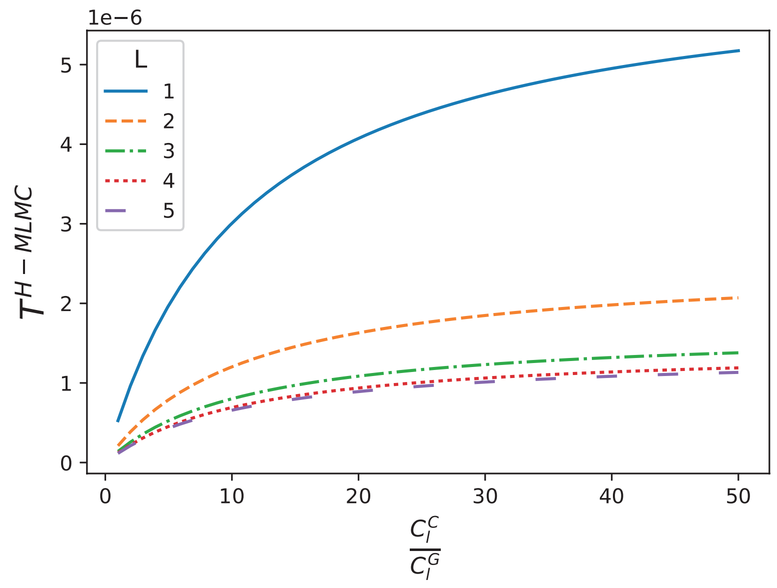

FIG. 6: Wall-clock time for one-, two-, three-, four-, and five-level H-MLMC for Example 1

is written in the Legion/Regent Programming system, which implements task-based parallelism [18,19]. The Regent programming language allows users to write sequential code, which the Legion runtime then executes in parallel with minimal user input. In addition, the Legion/Regent system can automatically generate CUDA code from Regent code and is easily portable between computer architectures. These characteristics make it simple to run Soleil-X in parallel on CPUs or GPUs on any computer architecture. We ran our test cases on Lassen [16].

Our test case duct has unit height, $L_y = 1$. We model a section of the duct with unit length, $L_x = 1$. The effect of particle adhesion is modeled through a stochastic heat flux boundary condition. Realization n of the stochastic heat flux, $\phi^{(n)}$, is a superposition of ten Gaussians:

$$\phi^{(n)}(x) = \Phi\left(1 + \sum_{i=1}^{10} A^{(n)} \exp\left(-\frac{1}{2}\left(\frac{x - \mu^{(n)}}{\sigma^{(n)}}\right)^2\right)\right), \tag{39}$$

where realization n of the stochastic parameters is $\sigma^{(n)} \sim \text{unif}(\sigma_{\min}, \sigma_{\max})$, $\mu \sim \text{unif}(2\sigma_{\max}, L_x - 2\sigma_{\max})$, and $A^{(n)} \sim \text{unif}(-\Phi, \Phi)$. The constants are $\Phi = 1000$, $\sigma_{\min} = 0.01$, and $\sigma_{\max} = 0.02$. The stochastic heat flux boundary condition is applied at $y = 0$ and a zero heat flux boundary condition is applied at $y = L_y$. Periodic boundary conditions are applied at the inflow, $x = 0$, and outflow, $x = L_x$. The flow is initialized from the steady solution for flow in the duct with zero heat flux boundary conditions, which has centerline velocity $U = 1$. Our quantity of interest is the temperature at $x = L_x$, $y = 0.1L_y$ at time $t = 1$, which is one flow-through time at the center of the duct. We compare the MLMC and H-MLMC estimators of our quantity of interest.

We use five levels, $L = 5$, for our MLMC and H-MLMC estimators. The levels are created by coarsening the grid. We estimate the CPU cost, GPU cost, and variance using ten CPU and ten GPU samples on each level. The CPU cost and GPU cost are the mean of the costs of these ten samples. To ensure these estimates are accurate while avoiding unnecessary sampling, in future work we could use an iterative method, such as the one used in [20]. The number of grid points, cost, and variance for each level are reported in Table 2. We also report the standard deviation of the CPU cost and GPU cost in Table 2. The sample allocation changes minimally due to the variation in the CPU and GPU cost across samples. For example, changing the CPU costs on all levels by one standard deviation or changing the CPU or GPU cost on a single level by one standard deviation changes the number of samples on a level by 0–3% and the ratios of wall-clock time for H-MLMC to that for MLMC by 0–1%. We take $\varepsilon^2 = 10^{-6}$.

We find that using all five levels, H-MLMC takes 14% as long as MLMC. Unlike for Example 1, the ratio of the H-MLMC wall-clock time to the MLMC wall-clock time, $T^{\text{H-MLMC}}/T^{\text{MLMC}}$, depends on L. Table 3 illustrates

TABLE 2: Example 2 grid point, cost, variance, and standard deviation, σ, values

l	**1**	**2**	**3**	**4**	**5**
grid points in x	64	128	128	256	512
grid points in y	32	64	128	128	128
C_ℓ^C/C_ℓ^G	1.0	1.9	3.4	6.4	12.1
C_ℓ^C	6.3×10^2	4.4×10^3	1.5×10^4	7.4×10^4	3.4×10^5
C_ℓ^G	6.4×10^2	2.2×10^3	4.6×10^3	1.2×10^4	2.8×10^4
$\sigma[C_\ell^C]$	3.7×10^1	1.7×10^2	2.5×10^2	5.1×10^2	5.4×10^2
$\sigma[C_\ell^G]$	5.5×10^1	1.6×10^2	2.7×10^2	7.7×10^2	1.2×10^3
$V[Q_\ell]$	1.8×10^{-4}	1.4×10^{-4}	1.1×10^{-4}	1.6×10^{-4}	1.3×10^{-4}
$V[Q_\ell - Q_{\ell-1}]$	N/A	1.6×10^{-5}	1.3×10^{-6}	1.1×10^{-6}	3.2×10^{-7}

TABLE 3: Example 2 ratio of wall-clock time for H-MLMC to that for MLMC, for MLMC and H-MLMC estimators constructed with the finest one, two, three, four, and five levels

L	**1**	**2**	**3**	**4**	**5**
$T^{\text{H-MLMC}}/T^{\text{MLMC}}$	0.52	0.37	0.23	0.17	0.14

the dependence on L. As more coarse levels are available, H-MLMC offers more improvement over MLMC. For example, H-MC takes 52% as long as MC and two-level H-MLMC takes 37% as long as two-level MLMC, whereas five-level H-MLMC takes 14% as long as five-level MLMC.

We show the optimal sample allocation for H-MLMC and MLMC in Table 4. Unlike in Example 1, Corollary 3 does not hold, so the total number of samples on each level, $N_\ell^C + N_\ell^G \neq (N_\ell^G)^{\text{MLMC}}$. Table 4 also shows the following properties of Theorem 2 hold. The ratio of C_ℓ^C/C_ℓ^G is different for each level, so only one level can have samples on both CPUs and GPUs (case 3 in Theorem 2). Here, level four has samples on both CPUs and GPUs. Since C_ℓ^C/C_ℓ^G increases monotonically, all levels coarser than level four (levels one, two, and three) can only have samples on CPUs (case 1). All levels finer than level four (level five) can only have samples on GPUs (case 2).

5. CONCLUSIONS

We present a variation on MLMC, H-MLMC, that uses the full computational power of heterogeneous computer architectures by running samples on two, rather than just one, type of processor. H-MLMC has a different optimal sample allocation than MLMC. We present the optimization problem for H-MLMC, demonstrate that it is convex, and derive the general solution. We show that the general solution cannot always be expressed analytically and is not always unique and introduce a computational procedure to find one of the possibly many optimal solutions.

We show that the optimal sample allocation is not unique on any level if and only if C_ℓ^C/C_ℓ^G is the same for all levels. In this case, the ratio of wall-clock time for H-MLMC to that for MLMC, $T^{\text{H-MLMC}}/T^{\text{MLMC}}$, depends only

TABLE 4: Example 2 optimal sample allocation for five-level H-MLMC, N_ℓ^C and N_ℓ^G, and MLMC, $(N_\ell^G)^{\text{MLMC}}$

l	**1**	**2**	**3**	**4**	**5**
N_ℓ^C	683	78	12	1	0
N_ℓ^G	0	0	0	4	2
$N_\ell^C + N_\ell^G$	683	78	12	5	2
$(N_\ell^G)^{\text{MLMC}}$	437	69	14	8	3

on C_ℓ^C/C_ℓ^G. If the ratio of CPU to GPU cost is not the same for all levels, we cannot, however, determine *a priori* whether the sample allocation is unique. In this case, $T^{\text{H-MLMC}}/T^{\text{MLMC}}$ depends not only on C_ℓ^C/C_ℓ^G, but also on the number of levels.

We demonstrate H-MLMC for two examples, one where C_ℓ^C/C_ℓ^G is the same for all levels and one where it is different on each level. When C_ℓ^C/C_ℓ^G is the same for all levels, H-MLMC takes 9–82% as long as MLMC, depending on the values of $C_\ell^C/C_\ell^G \in [5, 50]$, regardless of the number of levels used. H-MLMC offers more benefit over MLMC for smaller values of C_ℓ^C/C_ℓ^G. In the limit of C_ℓ^C/C_ℓ^G becoming large, the time to solution for H-MLMC approaches that for MLMC, since negligible benefit can be gained by using CPUs in addition to GPUs when the CPUs are infinitely slower than GPUs.

In our second example we consider the effect of a stochastic heat flux boundary condition on channel flow to inform the Stanford PSAAP II study on concentrated solar energy receivers. In this problem, C_ℓ^C/C_ℓ^G is different on each level. We find that H-MLMC takes 14–52% as long as MLMC, depending on the number of levels used. H-MLMC offers more benefit over MLMC when more levels are used.

ACKNOWLEDGMENTS

This research was supported in part by the U.S. Department of Energy Computational Science Graduate Fellowship under Grant No. DE SC0019323. Christiane Adcock is supported in part by a graduate fellowship award from Knight-Hennessy Scholars at Stanford University.

REFERENCES

1. Peherstorfer, B., Willcox, K., and Gunzburger, M., Survey of Multifidelity Methods in Uncertainty Propagation, Inference, and Optimization, *SIAM Rev.*, **60**(3):550–591, 2018.
2. Cliffe, K.A., Giles, M.B., Scheichl, R., and Teckentrup, A.L., Multilevel Monte Carlo Methods and Applications to Elliptic PDEs with Random Coefficients, *Comput. Visualization Sci.*, **14**(1):3–15, 2011.
3. Peherstorfer, B., Willcox, K., and Gunzberger, M., Optimal Model Management for Multifidelity Monte Carlo Estimation, *SIAM J. Sci. Comput.*, **38**(5):A3163–A3194, 2016.
4. Gorodetsky, A.A., Geraci, G., Eldred, M.S., and Jakeman, J.D., A Generalized Approximate Control Variate Framework for Multifidelity Uncertainty Quantification, *J. Comput. Phys.*, **408**:109257, 2020.
5. Geraci, G., Menhorn, F., Huan, X., Safta, C., Marzouk, Y.M., Najm, H.N., and Eldred, M.S., Progress in Scramjet Design Optimization under Uncertainty Using Simulations of the HIFiRE Direct Connect Rig, *AIAA SciTech Forum*, 2019.
6. Qian, E., Peherstorfer, B., Malley, D., Vesselinov, V., and Willox, K., Multifidelity Monte Carlo Estimation of Variance and Sensitivity Indices, *SIAM/ASA J. Uncertainty Quantif.*, **6**:683–706, 2018.
7. Krumscheid, S., Nobile, F., and Pisaroni, M., Quantifying Uncertain System Outputs via the Multilevel Monte Carlo Method. Part I: Central Moment Estimation, *J. Comput. Phys.*, **414**:109466, 2020.
8. Giles, M.B., Nagapetyan, T., and Ritter, K., Multilevel Monte Carlo Approximation of Distribution Functions and Densities, *SIAM/ASA J. Uncertainty Quantif.*, **3**(1):267–295, 2015.
9. Krumscheid, S. and Nobile, F., Multilevel Monte Carlo Approximation of Functions, *SIAM/ASA J. Uncertainty Quantif.*, **6**(3):1256–1293, 2018.
10. Top500, November 2019, accessed December 2, 2019, from http://top500.org/lists/top500/2019/11/.
11. Xu, X.G., Lui, T., Su, L., Du, X., Riblett, M., Ji, W., Gu, D., Carothers, C.D., Shephard, M.S., Brown, F.B., Kalra, M.K., and Liu, B., ARCHER, a New Monte Carlo Software Tool for Emerging Heterogeneous Computing Environments, *Ann. Nucl. Energy*, **82**:2–9, 2015.
12. Srinivasan, A., Parallel and Distributed Computing Issues in Pricing Financial Derivatives through Quasi Monte Carlo, in *Proc. of 16th International Parallel and Distributed Processing Symposium*, pp. 6–11, 2002.
13. Tse, A.H., Thomas, D.B., Tsoi, K., and Luk, W., Dynamic Scheduling Monte-Carlo Framework for Multi-Accelerator Heterogeneous Clusters, in *Proc. of Int. Conf. on Field-Programmable Technology*, pp. 233–240, 2010.
14. Giles, M.B., Multilevel Monte Carlo Codes for 2015 Acta Numerica Article, accessed December 2, 2019, from http://people.maths.ox.ac.uk/gilesm/acta/.

15. Jofre, L., Papadakis, M., Roy, P.T., Aiken, A., and Iaccarino, G., Multifidelity Modeling of Irradiated Particle-Laden Rurbulence Subject to Uncertainty, *Int. J. Uncertainty Quantif.*, **10**(6):499–514, 2020.
16. Lawrence Livermore National Laboratory, Lassen, 2019, accessed December 2, 2019, from http://hpc.llnl.gov/hardware/platforms/lassen.
17. Torres, H., Papadakis, M., Jofre, L., Lee, W., Aiken, A., and Iaccarino, G., Soleil-X: Turbulence, Particles, and Radiation in the Regent Programming Language, in *SC'19: Proc. of Int. Conf. on High Performance Computing, Networking, Storage, and Analysis*, 2019.
18. Slaughter, E., Lee, W., Treichler, S., Bauer, M., and Aiken, A., Regent: A High-Productivity Programming Language for HPC with Logical Regions, in *SC'15: Proc. of Int. Conf. on High Performance Computing, Networking, Storage, and Analysis*, 2015.
19. Bauer, M., Treichler, S., and Aiken, A., Legion: Expressing Locality and Independence with Logical Regions, in *SC'12: Proc. of Int. Conf. on High Performance Computing, Networking, Storage, and Analysis*, 2012.
20. Giles, M.B., Multilevel Monte Carlo Methods, *Acta Numer.*, **24**:259–328, 2015.

APPENDIX A. OPTIMAL SAMPLE ALLOCATION FOR H-MLMC

We seek to find a sample allocation, N_ℓ^C and $N_\ell^G\ \forall\ell = 1, 2, ..., L$, that satisfies Eq. (9). As shown in Section 2, a sample allocation satisfies Eq. (9) if it satisfies the KKT conditions, Eqs. (17)–(27).

Equation (17) is equivalent to

$$-\lambda_1\left(\frac{V_\ell}{(N_\ell^C+N_\ell^G)^2}\right)+\frac{\lambda_2 C_\ell^C}{M^C}-\mu_\ell^C=0\ \ \forall\ell=1,2,...,L, \tag{A.1}$$

$$-\lambda_1\left(\frac{V_\ell}{(N_\ell^C+N_\ell^G)^2}\right)+\frac{\lambda_3 C_\ell^G}{M^G}-\mu_\ell^G=0\ \ \forall\ell=1,2,...,L, \tag{A.2}$$

$$1-\lambda_2-\lambda_3=0. \tag{A.3}$$

By Eqs. (24) and (A.1),

$$-\lambda_1\frac{V_\ell}{(N_\ell^C+N_\ell^G)^2}+\frac{\lambda_2 C_\ell^C}{M^C}=\mu_\ell^C\geq 0\ \ \forall\ell=1,2,...,L. \tag{A.4}$$

Similarly, by Eqs. (25) and (A.2),

$$-\lambda_1\frac{V_\ell}{(N_\ell^C+N_\ell^G)^2}+\frac{\lambda_3 C_\ell^G}{M^G}=\mu_\ell^G\geq 0\ \ \forall\ell=1,2,...,L. \tag{A.5}$$

By Eqs. (21) and (A.4),

$$-\lambda_1\frac{V_\ell}{(N_\ell^C+N_\ell^G)^2}+\frac{\lambda_2 C_\ell^C}{M^C}>0\ \text{ iff }\ N_\ell^C=0\ \ \forall\ell=1,2,...,L. \tag{A.6}$$

Similarly, by Eqs. (22) and (A.5),

$$-\lambda_1\frac{V_\ell}{(N_\ell^C+N_\ell^G)^2}+\frac{\lambda_3 C_\ell^G}{M^G}>0\ \text{ iff }\ N_\ell^G=0\ \ \forall\ell=1,2,...,L. \tag{A.7}$$

N_ℓ^C and N_ℓ^G cannot both be zero; otherwise $V_\ell/(N_\ell^C+N_\ell^G)$ in Eq. (11) is undefined, making Eq. (11) unsatisfiable. Therefore, at least one of Eqs. (A.4) and (A.5) must equal zero. Whether Eq. (A.4), or Eq. (A.5), or both Eqs. (A.4) and (A.5) are zero depends on whether $(\lambda_3/\lambda_2)(>,=,<)(C_\ell^C M^G/C_\ell^G M^C)$. For each of these three cases, we can find N_ℓ^C and N_ℓ^G as follows:

Case 1: $\frac{\lambda_3}{\lambda_2} > \frac{C_\ell^C M^G}{C_\ell^G M^C}$

$$\lambda_3 C_\ell^G M^C > \lambda_2 C_\ell^C M^G. \tag{A.8}$$

By Eqs. (A.4), (A.5), and (A.8),

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_3 C_\ell^G M^C > -\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_2 C_\ell^C M^G \geq 0. \tag{A.9}$$

Since at least one of Eqs. (A.4) and (A.5) must be zero, by Eq. (A.9),

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_2 C_\ell^C M^G = 0, \tag{A.10}$$

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_3 C_\ell^G M^C > 0. \tag{A.11}$$

By Eqs. (A.7) and (A.11),

$$N_\ell^G = 0. \tag{A.12}$$

By Eqs. (A.8), (A.10), and (A.12),

$$N_\ell^C + N_\ell^G = N_\ell^C = \sqrt{\frac{\lambda_1 V_\ell}{\lambda_2 C_\ell^C M^G}} = \sqrt{\max\left(\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}, \frac{\lambda_1}{\lambda_3}\frac{V_\ell}{C_\ell^G M^C}\right)}. \tag{A.13}$$

Case 2: $\frac{\lambda_3}{\lambda_2} < \frac{C_\ell^C M^G}{C_\ell^G M^C}$

By analogous reasoning to case 1,

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_2 C_\ell^C M^G > 0, \tag{A.14}$$

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_3 C_\ell^G M^C = 0, \tag{A.15}$$

$$N_\ell^C = 0, \tag{A.16}$$

$$N_\ell^C + N_\ell^G = N_\ell^G = \sqrt{\frac{\lambda_1 V_\ell}{\lambda_3 C_\ell^G M^C}} = \sqrt{\max\left(\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}, \frac{\lambda_1}{\lambda_3}\frac{V_\ell}{C_\ell^G M^C}\right)}. \tag{A.17}$$

Case 3: $\frac{\lambda_3}{\lambda_2} = \frac{C_\ell^C M^G}{C_\ell^G M^C}$

$$\lambda_3 C_\ell^G M^C = \lambda_2 C_\ell^C M^G. \tag{A.18}$$

By Eqs. (A.4), (A.5), and (A.18):

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_3 C_\ell^G M^C = -\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_2 C_\ell^C M^G \geq 0. \tag{A.19}$$

Since at least one of Eqs. (A.4) and (A.5) must be zero, by Eqs. (26), (27), and (A.19):

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_2 C_\ell^C M^G = 0, \tag{A.20}$$

$$-\lambda_1 \frac{V_\ell}{(N_\ell^C + N_\ell^G)^2} + \lambda_3 C_\ell^G M^C = 0, \tag{A.21}$$

$$N_\ell^C > 0, \tag{A.22}$$

$$N_\ell^G > 0. \tag{A.23}$$

By Eqs. (A.20) and (A.21),

$$N_\ell^C + N_\ell^G = \sqrt{\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}} = \sqrt{\frac{\lambda_1}{\lambda_3}\frac{V_\ell}{C_\ell^G M^C}} = \sqrt{\max\left(\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}, \frac{\lambda_1}{\lambda_3}\frac{V_\ell}{C_\ell^G M^C}\right)} \tag{A.24}$$

Equation (A.24) does not uniquely determine N_ℓ^C and N_ℓ^G. By Eqs. (26), (27), and (A.24),

$$N_\ell^C \in \left[0, \sqrt{\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}}\right], \tag{A.25}$$

$$N_\ell^G = \sqrt{\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}} - N_\ell^C. \tag{A.26}$$

End Cases.

For all three cases, by Eqs. (A.13), (A.17), and (A.24),

$$N_\ell^C + N_\ell^G = \sqrt{\max\left(\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}, \frac{\lambda_1}{\lambda_3}\frac{V_\ell}{C_\ell^G M^C}\right)} = \sqrt{\lambda_1 V_\ell}\sqrt{\max\left(\frac{1}{\lambda_2 C_\ell^C M^G}, \frac{1}{\lambda_3 C_\ell^G M^C}\right)}. \tag{A.27}$$

By Eqs. (A.3), (A.20), and (A.21), and noting all costs are greater than zero,

$$\lambda_2 = \frac{1}{1+\alpha} > 0, \tag{A.28}$$

$$\lambda_3 = \frac{\alpha}{1+\alpha} > 0, \tag{A.29}$$

$$\alpha = \begin{cases} \dfrac{C_{l^*}^C M^G}{C_{l^*}^G M^C} \text{ if some level } l^* \text{ satisfies case 3} \\ \in \left[\min_{\ell=1,2,\ldots,L} \dfrac{C_\ell^C M^G}{C_\ell^G M^C}, \max_{\ell=1,2,\ldots,L} \dfrac{C_\ell^C M^G}{C_\ell^G M^C}\right] \text{ otherwise} \end{cases}. \tag{A.30}$$

By Eq. (A.27), and noting that all costs, variances, and sample sizes are greater than zero,

$$\lambda_1 = \frac{(N_\ell^C + N_\ell^G)^2}{V_\ell \max\big(1/(\lambda_2 C_\ell^C M^G), 1/(\lambda_3 C_\ell^G M^C)\big)} > 0 \tag{A.31}$$

Equations (A.28), (A.29), and (A.31) prove that λ_1, λ_2, and $\lambda_3 > 0$, as used in Section 3. By Eqs. (18) and (A.31),

$$\sum_{\ell=1}^{L} \frac{V_\ell}{N_\ell^C + N_\ell^G} = \varepsilon^2. \tag{A.32}$$

By Eqs. (A.27) and (A.32),

$$\lambda_1 = \left(\frac{1}{\varepsilon^2}\sum_{\ell=1}^{L}\sqrt{\frac{V_\ell}{\max\big(1/(\lambda_2 C_\ell^C M^G), 1/(\lambda_3 C_\ell^G M^C)\big)}}\right)^2. \tag{A.33}$$

By Eqs. (19), (20), (A.28), and (A.29),

$$\frac{1}{M^C}\sum_{\ell=1}^{L}\big(C_\ell^C N_\ell^C\big) = \frac{1}{M^G}\sum_{\ell=1}^{L}\big(C_\ell^G N_\ell^G\big). \tag{A.34}$$

We have now proved Theorem 2. Equations (32)–(38) correspond to Eqs. (A.34), (A.32), (A.33), (A.3), (A.12), and (13), (A.16) and (A.17), and (A.25) and (A.26), respectively.

As explained in Section 3, the optimal sample allocation cannot be found analytically, but can be found using the iterative bisection method of Algorithm 1. Step 5 of Algorithm 1 requires additional reasoning.

N_ℓ^C and N_ℓ^G for all levels that fall into cases 1 or 2 have already been computed in Step 4. These levels are in S, where

$$S = \left\{\ell \,\middle|\, \frac{C_\ell^C M^G}{C_\ell^G M^C} \neq \frac{\lambda_3}{\lambda_2}\right\}.$$

In Step 5 we first consider some level $\ell \notin S$. ℓ falls into case 3, so N_ℓ^C and N_ℓ^G are not fully defined.

Case A:

$$\frac{1}{M^C}\sum_{\ell\in S}\big(C_\ell^C N_\ell^C\big) = \frac{1}{M^G}\sum_{\ell\in S}\big(C_\ell^G N_\ell^G\big).$$

To keep the CPU and GPU cost balanced,

$$N_\ell = \sqrt{\frac{\lambda_1}{\lambda_2}\frac{V_\ell}{C_\ell^C M^G}} = \sqrt{\frac{\lambda_1}{\lambda_3}\frac{V_\ell}{C_\ell^G M^C}}, \tag{A.35}$$

$$N_\ell^C = \frac{N_\ell C_\ell^G M^C}{C_\ell^C M^G + C_\ell^G M^C}, \tag{A.36}$$

$$N_\ell^G = \frac{N_\ell C_\ell^C M^G}{C_\ell^C M^G + C_\ell^G M^C}. \tag{A.37}$$

Cases B.i. and B.ii.:

$$\frac{1}{M^C}\sum_{\ell\in S}\big(C_\ell^C N_\ell^C\big) > \frac{1}{M^G}\sum_{\ell\in S}\big(C_\ell^G N_\ell^G\big), \tag{A.38}$$

$$N_{\ell,\text{diff}} = \frac{M^G}{C_\ell^G}\left(\frac{1}{M^C}\sum_{\ell\in S}\big(C_\ell^C N_\ell^C\big) - \frac{1}{M^G}\sum_{\ell\in S}\big(C_\ell^G N_\ell^G\big)\right).$$

If we could add $N_{l,\text{diff}}$ samples to N_ℓ^G then the CPU and GPU work would be balanced.

Case B.i.:

$$N_{\ell,\text{diff}} \geq N_\ell$$

Adding $N_{l,\text{diff}}$ samples to N_ℓ^G would violate the condition that $N_\ell^C \geq 0$. Therefore, we assign as much work as possible on this level to the GPU while ensuring $N_\ell^C \geq 0$:

$$N_\ell^C = 0, \tag{A.39}$$

$$N_\ell^G = N_\ell. \tag{A.40}$$

Case B.ii.:

Equation (77) still holds but $N_{\ell,\text{diff}} < N_\ell$.

Now we can balance the CPU and GPU work by assigning $N_{\ell,\text{diff}}$ samples to N_ℓ^G. We split the remaining $N_\ell - N_{\ell,\text{diff}}$ samples between the CPU and GPU to keep the CPU and GPU work balanced:

$$N_\ell^C = (N_\ell - N_{l,\text{diff}}) \frac{C_\ell^G M^C}{C_\ell^C M^G + C_\ell^G M^C}, \tag{A.41}$$

$$N_\ell^G = N_{\ell,\text{diff}} + (N_\ell - N_{\ell,\text{diff}}) \frac{C_\ell^C M^G}{C_\ell^C M^G + C_\ell^G M^C}. \tag{A.42}$$

Cases C.i. and C.ii.:

$$\frac{1}{M^C} \sum_{\ell \in S} \big(C_\ell^C N_\ell^C\big) < \frac{1}{M^G} \sum_{\ell \in S} \big(C_\ell^G N_\ell^G\big), \tag{A.43}$$

$$N_{\ell,\text{diff}} = \frac{M^C}{C_\ell^C} \left(\frac{1}{M^G} \sum_{\ell \in S} \big(C_\ell^G N_\ell^G\big) - \frac{1}{M^C} \sum_{\ell \in S} \big(C_\ell^C N_\ell^C\big) \right).$$

Case C.i.:

$$N_{\ell,\text{diff}} \geq N_\ell.$$

By similar reasoning as in case B.i.,

$$N_\ell^C = N_\ell, \tag{A.44}$$

$$N_\ell^G = 0. \tag{A.45}$$

Case C.ii.:

Equation (A.43) still holds but $N_{\ell,\text{diff}} < N_\ell$.

By similar reasoning as in case B.ii.,

$$N_\ell^C = N_{\ell,\text{diff}} + (N_\ell - N_{\ell,\text{diff}}) \frac{C_\ell^G M^C}{C_\ell^C M^G + C_\ell^G M^C}, \tag{A.46}$$

$$N_\ell^G = (N_\ell - N_{l,\text{diff}}) \frac{C_\ell^C M^G}{C_\ell^C M^G + C_\ell^G M^C}. \tag{A.47}$$

We add ℓ to S. If $\exists \ell \notin S$, we repeat this step (Step 5).