Eidgenössische Technische Hochschule Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Static load balancing for multi-level Monte Carlo finite volume solvers

J. Šukys, S. Mishra and Ch. Schwab

Research Report No. 2011-32
May 2011

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Static load balancing for multi-level Monte Carlo finite volume solvers

Jonas Šukys, Siddhartha Mishra, and Christoph Schwab

ETH Zürich, Switzerland,
{jonas.sukys,smishra,schwab}@sam.math.ethz.ch.

Abstract. The Multi-Level Monte Carlo finite volumes (MLMC-FVM) algorithm was shown to be a robust and fast solver for uncertainty quantification in the solutions of multi-dimensional systems of stochastic conservation laws. A novel load balancing procedure is used to ensure scalability of the MLMC algorithm on massively parallel hardware. We describe this procedure together with other arising challenges in great detail. Finally, numerical experiments in multi-dimensions showing strong and weak scaling of our implementation are presented.

Keywords: uncertainty quantification, conservation laws, multi-level Monte Carlo, finite volumes, static load balancing, linear scaling.

Acknowledgments. This work is performed under ETH interdisciplinary research grant CH1-03 10-1. CS acknowledges partial support by the European Research Council under grant ERC AdG 247277 - STAHDPDE. JŠ is grateful to Stefan Pauli and Peter Arbenz for their contributions.

1 Introduction

A number of problems in physics and engineering are modeled in terms of systems of conservation laws:

$$\begin{cases} \mathbf{U}_t + \operatorname{div}(\mathbf{F}(\mathbf{U})) = \mathbf{S}(\mathbf{x}, \mathbf{U}), \\ \mathbf{U}(\mathbf{x}, 0) = \mathbf{U}_0(\mathbf{x}), \end{cases} \quad \forall (\mathbf{x}, t) \in \mathbb{R}^d \times \mathbb{R}_+. \tag{1}$$

Here, $\mathbf{U} : \mathbb{R}^d \to \mathbb{R}^m$ denotes the vector of conserved variables, $\mathbf{F} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^{m \times d}$ is the collection of directional flux vectors and $\mathbf{S} : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^m$ is the source term. The partial differential equation is augmented with initial data $\mathbf{U}_0$.

Examples for conservation laws include the shallow water equations of oceanography, the Euler equations of gas dynamics, the Magnetohydrodynamics (MHD) equations of plasma physics and the equations of non-linear elasticity.

As the equations are non-linear, analytical solution formulas are only available in very special situations. Consequently, numerical schemes such as Finite Volume methods [6] are required for the study of systems of conservation laws.

Existing numerical methods for approximating (1) require the initial data $\mathbf{U}_0$ and source $\mathbf{S}$ as the input. However, in most practical situations, it is not

possible to measure this input precisely. This uncertainty in the inputs for (1) propagates to the solution, leading to the *stochastic* system of conservation laws:

$$\begin{cases} \mathbf{U}(\mathbf{x},t,\omega)_t + \operatorname{div}(\mathbf{F}(\mathbf{U}(\mathbf{x},t,\omega))) = \mathbf{S}(\mathbf{x},\omega), \\ \mathbf{U}(\mathbf{x},0,\omega) = \mathbf{U}_0(\mathbf{x},\omega), \end{cases} \quad \mathbf{x}\in\mathbb{R}^d, \quad t>0, \quad \forall \omega\in\Omega. \tag{2}$$

where $(\Omega,\mathcal{F},\mathbb{P})$ is a complete probability space and initial data $\mathbf{U}_0$ with the source term $\mathbf{S}$ are random fields [1,2]. The solution is also realized as a random field; its statistical moments like the expectation and variance are the quantities of interest. An estimate of expectation can be obtained by the so-called Monte Carlo finite volume method (MC-FVM) consisting of the following three steps:

1. **Sample:** We draw M independent identically distributed (i.i.d.) initial data and source samples $\{\mathbf{U}_0^i,\mathbf{S}_0^i\}$ with $i=1,2,\ldots,M$ from the random fields $\{\mathbf{U}_0,\mathbf{S}_0\}$ and approximate these by piecewise constant cell averages.
2. **Solve:** For each realization $\{\mathbf{U}_0^i,\mathbf{S}_0^i\}$, the underlying conservation law (1) is solved numerically by the finite volume method [6,3,5]. We denote the FVM solutions by $\mathbf{U}_{\mathcal{T}}^{i,n}$, i.e. by cell averages $\{\mathbf{U}_K^{i,n} : K\in\mathcal{T}\}$ at the time level t^n.
3. **Estimate Statistics:** We estimate the expectation of the random solution field with the sample mean (ensemble average) of the approximate solution:

$$E_M[\mathbf{U}_{\mathcal{T}}^n] := \frac{1}{M}\sum_{i=1}^{M}\mathbf{U}_{\mathcal{T}}^{i,n}, \quad M=\mathcal{O}(\Delta x^{-\frac{s}{2}}). \tag{3}$$

The error of the MC-FVM asymptotically scales [2] as $(\text{Work})^{\frac{-s}{(d+1+2s)}}$, making the MC-FVM method computationally infeasible when high accuracy is needed.

The multi-level Monte Carlo finite volume method (MLMC-FVM) was recently proposed in [2,1]. The key idea behind MLMC-FVM is to simultaneously draw MC samples on a hierarchy of nested grids. There are four main steps:

1. **Nested meshes:** Consider *nested* triangulations $\{\mathcal{T}_\ell\}_{\ell=0}^{\infty}$ of the spatial domain with corresponding mesh widths Δx_ℓ that satisfy:

$$\Delta x_\ell = \Delta x(\mathcal{T}_\ell) = \sup\{\operatorname{diam}(K) :\ K\in\mathcal{T}_\ell\} = \mathcal{O}(2^{-\ell}\Delta x_0), \quad \ell\in\mathbb{N}_0, \tag{4}$$

   where Δx_0 - mesh width of the coarsest resolution at the lowest level $\ell=0$.
2. **Sample:** For each level of resolution $\ell\in\mathbb{N}_0$, we draw M_ℓ independent identically distributed (i.i.d) samples $\{\mathbf{U}_{0,\ell}^i,\mathbf{S}_{0,\ell}^i\}$ with $i=1,2,\cdots,M_\ell$ from the random fields $\{\mathbf{U}_0,\mathbf{S}_0\}$ and approximate these by cell averages.
3. **Solve:** For each resolution level ℓ and each realization $\{\mathbf{U}_{0,\ell}^i,\mathbf{S}_{0,\ell}^i\}$, the underlying balance law (1) is solved by the finite volume method [6,3,5] with mesh width Δx_ℓ; denote solutions by $\mathbf{U}_{\mathcal{T}_\ell}^{i,n}$ at the time t^n and mesh level ℓ.
4. **Estimate solution statistics:** Fix some positive integer $L<\infty$ corresponding to the highest level. We estimate the expectation of the random solution field with the following estimator:

$$E^L[\mathbf{U}(\cdot,t^n)] := \sum_{\ell=0}^{L} E_{M_\ell}[\mathbf{U}_{\mathcal{T}_\ell}^n - \mathbf{U}_{\mathcal{T}_{\ell-1}}^n], \tag{5}$$

with E_{M_ℓ} being the MC estimator defined in (3) for the level ℓ. Higher statistical moments can be approximated analogously (see, e.g., [2]).

Authors in [2] provide error vs. computational work estimate leading to the following required number of samples on each discretization level ℓ to equilibrate the statistical and spatio-temporal discretization errors in (5):

$$M_\ell = \mathcal{O}(2^{2(L-\ell)s}). \tag{6}$$

Notice that (6) implies that the largest number of MC samples is required on the coarsest mesh level $\ell = 0$, whereas only a small fixed number of MC samples are needed on the finest discretization levels.

The corresponding error vs. work estimate for MLMC-FVM is given by [1,2],

$$\text{error} \lesssim \begin{cases} (\text{Work})^{-s/(d+1)} & \text{if } s < (d+1)/2, \\ \left(\frac{\text{Work}}{\log(\text{Work})}\right)^{-s/(d+1)} & \text{if } s = (d+1)/2. \end{cases} \tag{7}$$

The above estimates show that the MLMC-FVM is superior to the MC-FVM. Furthermore, if convergence rate s of the FVM solver satisfies $s < (d+1)/2$ then this estimate is exactly of the same order as the estimate for the *deterministic* finite volume scheme. For the same error, the MLMC-FVM was shown to be considerably faster than the MC-FVM [1,2]; in particular, at the relative error level of 1%, the speed up reached approximately two orders of magnitude [1,2].

2 Highly scalable implementation of MLMC-FVM

MLMC-FVM is *non-intrusive* as any standard FVM code can be used in step 3. Furthermore, MLMC-FVM is amenable to *efficient parallelization* as data from different grid resolutions and samples only interacts in step 4. Select a nested hierarchy of triangulations in step 1 is straightforward for any parallel architecture. In step 2, we draw samples for $\{\mathbf{U}_0, \mathbf{S}_0\}$ with a given probability distribution. Here, a robust random number generator (RNG) is needed.

2.1 Robust pseudo random number generation

Random number generation becomes a very sensitive part of Monte Carlo type algorithms on massively parallel architectures. Inconsistent seeding and insufficient period length of the RNG might cause correlations in presumably i.i.d. draws which might potentially lead to biased solutions such as in Figure 5 of [1].

Such spurious correlations are due to two factors: firstly, large number of MC samples requires longer period and hence larger buffer of the RNG. Secondly, the seeding of the buffer for each core must preserve statistical independence.

For the numerical simulations reported below, we used the `WELL`-series [9] of pseudo random number generators, which were designed with particular attention towards large periods and good equidistribution. In particular, the RNG

`WELL512a` was used; we found `WELL512a` to have a sufficiently large period $2^{512}-1$ and to be reasonably efficient (33 CPU sec for 10^9 draws). We emphasize that there are plenty of alternatives to `WELL512a` with even longer periods (which, however, use more memory). To name a few: `WELL1024a` with period $2^{1024}-1$, takes 34 sec and `WELLRNG44497` with period $2^{44497}-1$ which takes 41 sec for 10^9 draws. The strategy to deal with seeding issues is described in subsection 2.4.

In step 3 of the MLMC-FVM algorithm, we solve the conservation law (2) for each draw of the initial data. This is performed with `ALSVID`. A massively parallel version of `ALSVID` has already been developed for deterministic problems; refer to [7] for further details. The parallelization paradigm for `ALSVID` is based on domain decomposition using `MPI` standard [12].

2.2 A priori estimates for computational work

The key issue in the parallel implementation of the solve steps (in the Step 3 of MLMC-FVM algorithm) is to distribute computational work evenly among the cores. The FVM algorithm consists of computing fluxes across all cell interfaces and then updating cell averages via the explicit stable time stepping routine [6]. The computational complexity of numerical flux approximations is given by an explicit algorithm and is of order equal to the number of cells in the mesh $\mathcal{T}$,

$$\mathrm{Work}_{\mathcal{T}}^{\mathrm{step}} = \mathrm{Work}^{\mathrm{step}}(\Delta x) = \mathcal{O}(N) = \mathcal{O}(\Delta x^{-d}), \tag{8}$$

where $N = \#\mathcal{T}$ denotes the number of cells and Δx denotes the mesh width of triangulation $\mathcal{T}$. To ensure stability of the FVM scheme, a CFL condition [6] is imposed on the time step size $\Delta t := t^{n+1} - t^n$, which forces

$$\Delta t = \mathcal{O}(\Delta x). \tag{9}$$

Hence, the computational work $\mathrm{Work}_{\mathcal{T}}^{\mathrm{det}}$ for *one* complete *deterministic* solve using the FVM method on the triangulation $\mathcal{T}$ with mesh width Δx is given by multiplying the work for one step (8) by the total number of steps $\mathcal{O}(\Delta t^{-1})$,

$$\mathrm{Work}_{\mathcal{T}}^{\mathrm{det}} = \mathrm{Work}_{\mathcal{T}}^{\mathrm{step}} \cdot \mathcal{O}(\Delta t) \overset{(9)}{=} \mathcal{O}(\Delta x^{-d}) \cdot \mathcal{O}(\Delta x^{-1}) = \mathcal{O}(\Delta x^{-(d+1)}). \tag{10}$$

In most explicit FVM schemes (e.g. Rusanov with (W)ENO and SSP-RK2, see [6]), all lower order terms $\mathcal{O}(\Delta x^{-d})$ in (10) are negligible, even when a very coarse mesh is used. Hence, we assume that (10) holds in a stricter sense,

$$\mathrm{Work}_{\mathcal{T}}^{\mathrm{det}} = K \Delta x^{-(d+1)}, \tag{11}$$

where constant K depends on the FVM that is used and on the time horizon $t > 0$, but does *not* depend on mesh width Δx. Finally, MC algorithm (3) combines multiple deterministic solves for a sequence of sample draws $\omega \in \Omega$. By $\mathrm{Work}_M(\Delta x)$ we denote the computational work needed for a MC-FVM algorithm performing M solves, each of complexity as in (11). Then,

$$\mathrm{Work}_M(\Delta x) = M \cdot \mathrm{Work}^{\mathrm{det}}(\Delta x) = M \cdot K \Delta x^{-(d+1)}. \tag{12}$$

Next, we describe our load balancing strategy needed for the Step 3.

2.3 Static load balancing

In what follows, we assume *a homogeneous computing environment* meaning that all cores are assumed to have identical CPUs and RAM per node, and equal bandwidth and latency to all other cores. There are 3 levels of parallelization: across mesh resolution levels, across MC samples and *inside* the deterministic solver using domain decomposition (see example in Figure 1). Domain decomposition is used only in the few levels with the finest mesh resolution. On these levels, the number of MC samples is small. However, these levels require most of the computational effort (unless $s = (d+1)/2$ in (7) holds).

For the finest level $\ell = L$ we *fix* the number of cores:

$$\underbrace{C_L}_{\#\text{ of cores}} = \underbrace{D_L}_{\#\text{ of subdomains}} \times \underbrace{P_L}_{\#\text{ of groups for MC samples}} \tag{13}$$

where for every level $0 \leq \ell \leq L$, D_ℓ denotes the number of subdomains and P_ℓ denotes the number of "samplers" - groups of cores, where every such group computes some portion of required M_ℓ Monte Carlo samples at level ℓ. We assume that each subdomain is computed on exactly one core and denote the total number of cores at level $0 \leq \ell \leq L$ by C_ℓ, i.e.

$$C_\ell = D_\ell P_\ell, \quad \forall \ell \in \{0, \ldots, L\}. \tag{14}$$

Since total computational work for $E_{M_\ell}[\mathbf{U}^n_{\mathcal{T}_\ell} - \mathbf{U}^n_{\mathcal{T}_{\ell-1}}]$ is then given by (12), i.e.

$$\text{Work}_\ell := \text{Work}_{M_\ell}(\Delta x_\ell) + \text{Work}_{M_\ell}(\Delta x_{\ell-1}), \tag{15}$$

the ratio of computational work for the remaining levels $\ell \in \{L-1, \ldots, 1\}$ is given recursively by inserting (6) into a priori work estimates (12):

$$\begin{aligned} \frac{\text{Work}_\ell}{\text{Work}_{\ell-1}} &= \frac{M_L 2^{2(L-\ell)s} K \left(\Delta x_\ell^{-(d+1)} + \Delta x_{\ell-1}^{-(d+1)} \right)}{M_L 2^{2(L-(\ell-1))s} K \left(\Delta x_{\ell-1}^{-(d+1)} + \Delta x_{\ell-2}^{-(d+1)} \right)} \\ &= \frac{2^{-2\ell s}}{2^{-2(\ell+1)s}} \frac{\Delta x_\ell^{-(d+1)} \left(1 + 2^{-(d+1)}\right)}{\Delta x_\ell^{-(d+1)} \left(2^{-(d+1)} + 2^{-2(d+1)}\right)} = 2^{d+1-2s}. \end{aligned} \tag{16}$$

For level $\ell = 0$, the term $\mathbf{U}^n_{\mathcal{T}_{-1}}$ in $E_{M_0}[\mathbf{U}^n_{\mathcal{T}_0} - \mathbf{U}^n_{\mathcal{T}_{-1}}]$ is known ($\equiv 0$), hence (16) provides a lower bound rather than an equality, i.e. $\text{Work}_0 \leq \text{Work}_1/(2^{d+1-2s})$.

Consequently, the positive integer parameters D_L and $P_L \leq M_L$ recursively determine the number of cores needed for each level $\ell < L$ via the relation

$$C_\ell = \left\lceil \frac{C_{\ell+1}}{2^{d+1-2s}} \right\rceil, \quad \forall \ell < L. \tag{17}$$

Notice, that the denominator 2^{d+1-2s} in (17) is a *positive integer* (a power of 2) provided $s \in \mathbb{N}/2$ and $s \leq (d+1)/2$ (which is *not* an additional constraint as it is also present in (7)). However, when $s < (d+1)/2$, we have:

$$2^{d+1-2s} \geq 2, \tag{18}$$

which (when L is large) leads to inefficient load distribution for levels $\ell \leq \ell^*$, where each successive level needs needs *less* than one core:

$$\ell^* := \min\{0 \leq \ell \leq L : C_{\ell+1} < 2^{d+1-2s}\}. \tag{19}$$

We investigate the amount of *total* computational work ($\text{Work}_{\{0,\dots,\ell^*\}}$) required for such "inefficient" levels $\ell \in \{0,\dots,\ell^*\}$:

$$\begin{aligned}\text{Work}_{\{0,\dots,\ell^*\}} :&= \sum_{\ell=0}^{\ell^*} \text{Work}_\ell \overset{(16)}{=} \sum_{\ell=0}^{\ell^*} \frac{\text{Work}_{\ell^*}}{2^{(d+1-2s)(\ell^*-\ell)}} \leq \sum_{\ell=-\infty}^{\ell^*} \frac{\text{Work}_{\ell^*}}{2^{(d+1-2s)(\ell^*-\ell)}} \\ &= \frac{\text{Work}_{\ell^*}}{1-(2^{d+1-2s})^{-1}} \overset{(18)}{\leq} \frac{\text{Work}_{\ell^*}}{1-\frac{1}{2}} = 2 \cdot \text{Work}_{\ell^*}.\end{aligned} \tag{20}$$

For the sake of simplicity, assume that P_L and D_L are nonnegative integer powers of 2. Under this assumption, definition (19) of ℓ^* together with recurrence relation (17) *without* rounding up ($\lceil\cdot\rceil$) implies that $C_{\ell^*} \leq 1/2$. Hence, total work estimate (20) for *all* levels $\ell \in \{0,\dots,\ell^*\}$ translates into an estimate for sufficient number of cores, which, instead of ℓ^*+1, turns out to be *only* 1:

$$\text{Work}_{\{0,\dots,\ell^*\}} \leq 2 \cdot \text{Work}_{\ell^*} \quad \longrightarrow \quad C_{\{0,\dots,\ell^*\}} \leq 2 \cdot \frac{1}{2} = 1. \tag{21}$$

The implementation of (21) (i.e. multiple levels per 1 core) is *essential* to obtain efficient and highly scalable parallelization of MLMC-FVM when $s < \frac{d+1}{2}$.

The example of static load distribution for MLMC-FVM algorithm using all three parallelization levels is given in Figure 1, where the parameters are set to:

$$L = 5, \quad M_L = 4, \quad d = 1, \quad s = \frac{1}{2}, \quad D_L = 2, \quad P_L = 4.$$

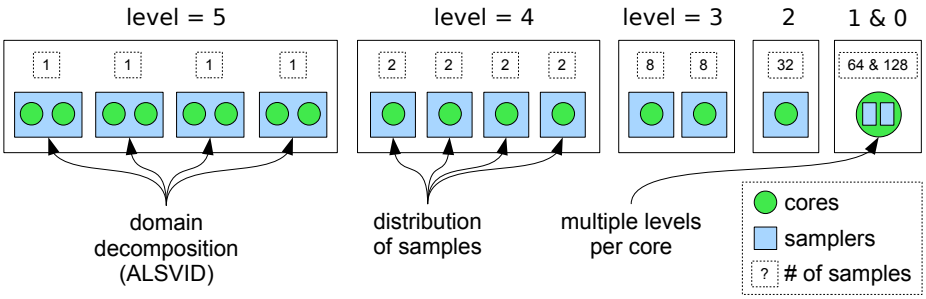

Fig. 1. Example for the static load distribution structure

Next, we describe our implementation of this load balancing using `C++` and `MPI`.

2.4 Implementation using C++ and MPI

In what follows, we assume that each MPI process is running on its own core. Simulation is divided into 3 main phases - initialization, simulation and data collection; key concepts for the implementation of the static load balancing algorithm for each phase is described below.

Phase 1 - Initialization:

– **MPI groups and communicators.** By default, message passing in `MPI` is done via the main communicator `MPI_COMM_WORLD` which connects *all* processes. Each process has a prescribed unique non-negative integer called *rank*. The process with the rank 0 is called *root*. Apart from `MPI_COMM_WORLD`, we use `MPI_Group_range_incl()` and `MPI_Comm_create()` to create sub-groups and corresponding inter-communicators, this way empowering message passing within particular *subgroups* of processes. Such communicators ease the use of collective reduction operations within some particular *subgroup* of processes. We implemented 3 types of communicators (see Figure 2):

1. **Domain** communicators connect processes within each sampler; these precesses are used for domain decomposition of one physical mesh.
2. **Sampler** communicators connect processes that work on the MC samples at the *same* mesh level.
3. **Level** communicators connect only the processes (across *all* levels) that are roots of *both* domain and sampler communicators,

where, analogously to `MPI_COMM_WORLD`, every process has a unique rank in each of the communicators 1-3; processes with rank 0 in domain communicators are called *domain roots*, in sampler communicators - *sampler roots*, and in level communicators - *level roots*. `MPI_COMM_WORLD` is used *only* in `MPI_Init()`, `MPI_Finalize()` and `MPI_Wtime()`. Figure 2 depicts all non-trivial communicators and roots for the example setup as in Figure 1.

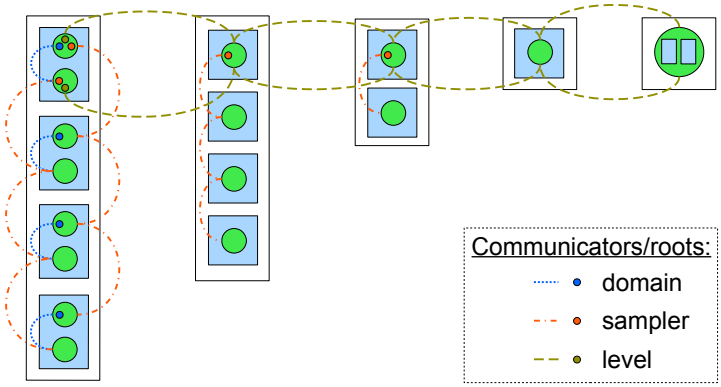

Fig. 2. Structure and root processes of the communicators for setup as in Figure 1

– **Seeding RNG.** To deal with the seeding issues mentioned in subsection 2.1, we *injectively* (i.e. one-to-one) map the *unique* rank (in `MPI_COMM_WORLD`) of each process that is root in *both* domain and sampler communicators to some corresponding element in the hardcoded array of prime numbers (seeds). Then these processes generate random vectors of real numbers needed to compute MC samples $\{\mathbf{U}^i_{0,\ell}, \mathbf{S}^i_{0,\ell}\}$ for the *entire* level ℓ. Afterwards, samples are scattered evenly using `MPI_Scatter()` via the sampler communicator; `MPI_IN_PLACE` is used to remove unnecessary memory overhead. Finally, samples are broadcast using `MPI_Bcast()` via the domain communicator.

Phase 2 - Simulation:

– **FVM solves for 2 mesh levels.** FVM solves of each sample are performed for $E_{M_\ell}[\mathbf{U}^n_{\mathcal{T}_\ell}]$ and for $E_{M_\ell}[\mathbf{U}^n_{\mathcal{T}_{\ell-1}}]$, and then combined into $E_{M_\ell}[\mathbf{U}^n_{\mathcal{T}_\ell} - \mathbf{U}^n_{\mathcal{T}_{\ell-1}}]$.
– **Inter-domain communication.** Cell values near interfaces of adjacent subdomains are exchanged asynchronously with `MPI_Isend()` and `MPI_Recv()`.

Phase 3 - Data collection and output:

– **MC estimator.** For each level, statistical estimates are collectively reduced with `MPI_Reduce()` into sampler roots using sampler communicators; then MC estimators (3) for mean and variance (see subsection 2.5) are obtained.
– **MLMC estimator.** MC estimators from different levels are finally combined via level communicators to level roots to obtain MLMC estimator (5).
– **Parallel data output.** Each process that is *both* sampler root and level root writes out final result. Hence, the number of parallel output files is equal to the number of subdomains on the finest mesh level.

This concludes the discussion of static load balancing and of step 3 of MLMC-FVM. In step 4, the results are combined to compute sample mean and variance.

2.5 Variance computation for parallel runs

A numerically *stable* serial variance computation algorithm (so-called "online") is given as follows [10]: set $\bar{u}^0 = 0$ and $\Phi^0 = 0$; then proceed recursively,

$$\bar{u}^i = \sum_{j=1}^{i} u^j/i, \quad \Phi^i := \sum_{j=1}^{i} (u^j - \bar{u}^i)^2 = \Phi^{i-1} + (u^i - \bar{u}^i)(u^i - \bar{u}^{i-1}). \tag{22}$$

Then, the unbiased mean and variance estimates are given by:

$$E_M[u] = \bar{u}^M, \quad \mathrm{Var}_M[u] = \Phi^M/(M-1). \tag{23}$$

For parallel architectures, assume we have 2 cores A and B each computing M_A and M_B ($M = M_A + M_B$) number of samples respectively. Then an *unbiased* estimate for mean and variance can be obtained by [11]

$$E_M[u] = \frac{M_A E_{M_A}[u] + M_B E_{M_B}[u]}{M}, \quad \mathrm{Var}_M[u] = \frac{\Phi^M}{M-1}, \tag{24}$$

$$\text{where:} \quad \Phi^M = \Phi^{M_A} + \Phi^{M_B} + \delta^2 \cdot \frac{M_A \cdot M_B}{M}, \quad \delta = E_{M_B}[u] - E_{M_A}[u]. \tag{25}$$

Then, for any finite number of cores formula (24) is applied recursively.

3 Efficiency and linear scaling in numerical simulations

The static load balancing algorithm was tested on a series of standard benchmarks for hyperbolic solvers. For detailed description of the setup, refer to [1] for Euler equations of gas dynamics, [1,3] for Magnetohydrodynamics (MHD) equations of plasma physics, and [4] for shallow water equations with randomly varying bottom topography. The runtime of all aforementioned simulations was measured by the so-called *wall-time*, accessible as `MPI_Wtime()` routine in `MPI2.0`. We define parallel efficiency as a fraction of *pure simulation* time over total time,

$$\text{efficiency} := \frac{(\text{cumulative wall-time}) - (\text{cumulative wall-time of MPI calls})}{\text{cumulative wall-time}}. \tag{26}$$

In Figure 3 we verify *strong scaling* (fixed discretization and sampling parameters while increasing #cores) and in Figure 4 we verify *weak scaling* (problem size is equivalent to #cores) of our implementation in 1d. In 2d simulations, we maintained such scaling upto 1023 cores at 97% efficiency. We believe that our parallelization algorithm will scale linearly for a much larger number of cores.

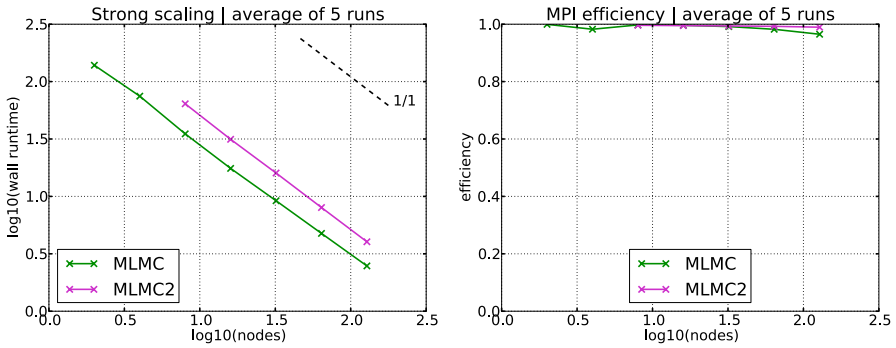

Fig. 3. Strong scaling. Domain decomposition method (DDM) is enabled from $10^{1.5}$ cores onwards (for MLMC only); its scalability is inferior to pure (ML)MC parallelization due to additional networking between sub-domain boundaries.

4 Conclusion

MLMC-FVM algorithm is superior to standard MC algorithms for uncertainty quantification in hyperbolic conservation laws, and yet, as most sampling algorithms, it still scales linearly w.r.t. number of uncertainty sources. Due to its non-intrusiveness, MLMC-FVM was efficiently parallelized for multi-core architectures. Strong and weak scaling of our implementation `ALSVID-UQ` [8] of the proposed static load balancing was verified on the high performance clusters [13,14] in multiple space dimensions. The suite of benchmarks included Euler equations of gas dynamics, MHD equations [1], and shallow water equations [4].

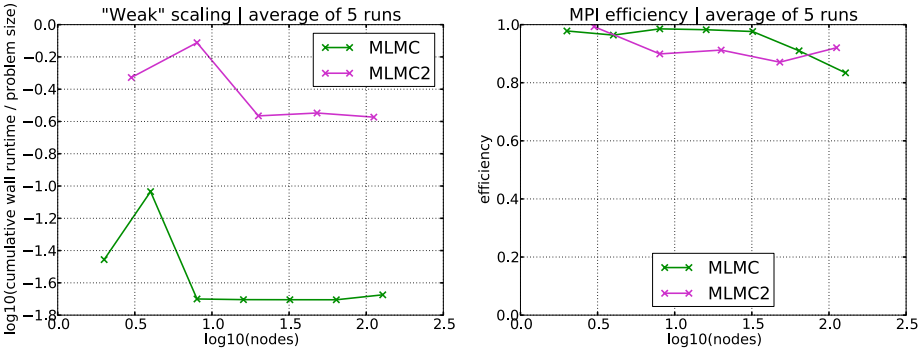

Fig. 4. Weak scaling. Analogously as in Figure 3, the slight deterioration in MLMC scaling from $10^{1.5}$ cores onwards could have been caused by inferior scaling of DDM.

References

1. S. Mishra, Ch. Schwab and J. Šukys. *Multi-level Monte Carlo finite volume methods for nonlinear systems of conservation laws in multi-dimensions.* Submitted, 2011. Available from: `http://www.sam.math.ethz.ch/reports/2011/02`.
2. S. Mishra and Ch. Schwab. *Sparse tensor multi-level Monte Carlo Finite Volume Methods for hyperbolic conservation laws with random initial data.* Preprint 2010. Available from `http://www.sam.math.ethz.ch/reports/2010/24`.
3. F. Fuchs, A. D. McMurry, S. Mishra, N. H. Risebro and K. Waagan. *Approximate Riemann solver based high-order finite volume schemes for the Godunov-Powell form of ideal MHD equations in multi-dimensions.* Comm. Comp. Phys., **9**:324-362, 2011.
4. S. Mishra, Ch. Schwab and J. Šukys. *Multi-level Monte Carlo finite volume methods for shallow water equations with uncertain topography in multi-dimensions.* In progress, 2011.
5. U.S. Fjordholm, S. Mishra, and E. Tadmor. *Well-balanced, energy stable schemes for the shallow water equations with varying topology.* Submitted, 2010. Available from `http://www.sam.math.ethz.ch/reports/2010/26`.
6. R.A. LeVeque. *Numerical Solution of Hyperbolic Conservation Laws.* Cambridge Univ. Press 2005.
7. ALSVID. Available from `http://folk.uio.no/mcmurry/amhd`.
8. ALSVID-UQ. Available from `http://mlmc.origo.ethz.ch/`.
9. P. L'Ecuyer and F. Panneton. *Fast Random Number Generators Based on Linear Recurrences Modulo 2.* ACM Trans. Math. Software, **32**:1-16, 2006.
10. B. P. Welford. *Note on a Method for Calculating Corrected Sums of Squares and Products.* Technometrics, **4**:419-420, 1962.
11. T. F. Chan, G. H. Golub. and R. J. LeVeque. *Updating Formulae and a Pairwise Algorithm for Computing Sample Variances.* STAN-CS-79-773, 1979.
12. *MPI: A Message-Passing Interface Standard.* Version 2.2, 2009. Available from: `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf`.
13. *Brutus,* ETH Zürich, `de.wikipedia.org/wiki/Brutus_(Cluster)`.
14. *Palu,* Swiss National Supercomputing Center (CSCS), Manno, `www.cscs.ch`.

Research Reports

No.	Authors/Title
11-32	*J. Šukys, S. Mishra and Ch. Schwab* Static load balancing for multi-level Monte Carlo finite volume solvers
11-31	*C.J. Gittelson, J. Könnö, Ch. Schwab and R. Stenberg* The multi-level Monte Carlo Finite Element Method for a stochastic Brinkman problem
11-30	*A. Barth, A. Lang and Ch. Schwab* Multi-level Monte Carlo Finite Element method for parabolic stochastic partial differential equations
11-29	*M. Hansen and Ch. Schwab* Analytic regularity and nonlinear approximation of a class of parametric semilinear elliptic PDEs
11-28	*R. Hiptmair and S. Mao* Stable multilevel splittings of boundary edge element spaces
11-27	*Ph. Grohs* Shearlets and microlocal analysis
11-26	*H. Kumar* Implicit-explicit Runge-Kutta methods for the two-fluid MHD equations
11-25	*H. Papasaika, E. Kokiopoulou, E. Baltsavias, K. Schindler and D. Kressner* Sparsity-seeking fusion of digital elevation models
11-24	*H. Harbrecht and J. Li* A fast deterministic method for stochastic elliptic interface problems based on low-rank approximation
11-23	*P. Corti and S. Mishra* Stable finite difference schemes for the magnetic induction equation with Hall effect
11-22	*H. Kumar and S. Mishra* Entropy stable numerical schemes for two-fluid MHD equations
11-21	*H. Heumann, R. Hiptmair, K. Li and J. Xu* Semi-Lagrangian methods for advection of differential forms
11-20	*A. Moiola* Plane wave approximation in linear elasticity
11-19	*C.J. Gittelson* Uniformly convergent adaptive methods for parametric operator equations