Parallel Computing 39 (2013) 638–651

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Hybrid parallel computing of minimum action method

Xiaoliang Wan [a,*], Guang Lin [b]

[a] Department of Mathematics and Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803, USA
[b] Pacific Northwest National Laboratory, Richland, WA 99352, USA

ARTICLE INFO

Article history:
Received 18 October 2012
Received in revised form 4 June 2013
Accepted 10 August 2013
Available online 4 September 2013

Keywords:
Random perturbation
Dynamical system
Minimum action method
Rare events
Spectral elements
Heterogeneous computing

ABSTRACT

In this work, we report a hybrid (MPI/OpenMP) parallelization strategy for the minimum action method recently proposed in [17]. For nonlinear dynamical systems, the minimum action method is a useful numerical tool to study the transition behavior induced by small noise and the structure of the phase space. The crucial part of the minimum action method is to minimize the Freidlin–Wentzell action functional. Due to the fact that the corresponding Euler–Lagrange equation is, in general, highly nonlinear and of high order, we solve the optimization problem directly instead of discretizing the Euler–Lagrange equation to provide a general but equivalent numerical framework. To enhance the efficiency of the minimum action method for general dynamical systems we consider parallel computing. In particular, we present a hybrid parallelization strategy based on MPI and OpenMP. Numerical results are presented to demonstrate the efficiency of the proposed parallelization strategy.

1. Introduction

Dynamical systems are often subject to random perturbations since noise is ubiquitous in nature. Even when these random perturbations have a small amplitude, they can produce a profound effect on the long time dynamics by inducing rare but important events. A large number of interesting phenomena in physics, chemistry and biology such as phase transitions, biological switches and chemical reactions, etc., are examples of such noise-induced rare events [11].

When the random perturbations are small, the Freidlin–Wentzell theory of large deviations provides a rigorous mathematical framework for us to understand how the transitions occur and how frequent they are. The transition pathways between metastable sets in a dynamical system often have a rather deterministic nature. As the noise amplitude decreases to zero, the events for successful transitions between metastable sets have a sharply peaked probability around a certain deterministic path that is least unlikely. Special features of such a path tell us crucial information about the mechanism of the transition. One class of examples that have been well studied for a long time are the gradient systems, for which the vector field is the gradient of a potential function. In gradient systems, the most probable transition path is the minimum energy path (MEP), which passes through the basin boundary between the stable states at some saddle points of index one [13]. For non-gradient systems we need to consider the action functional instead of the energy, which is the central object to the Freidlin–Wentzell theory. The minimizer of the action functional provides the most probable transition path; the minimum of the action functional provides an estimate of the probability and the rate of occurrence of the transition. Thus an important practical task is to compute the minimum and minimizer of the action functional. A large number of algorithms have been designed for gradient systems. Some popular algorithms include the string method [2,4], nudged elastic band method [10], eigenvector-following-type method (e.g., [1]) as well as the dimer method [9], which usually take advantage of the fact that

* Corresponding author.
E-mail addresses: xlwan@math.lsu.edu (X. Wan), guang.lin@pnl.gov (G. Lin).

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.08.004

in gradient systems the transition paths are always parallel to the drift term of the stochastic differential equation. For non-gradient systems, we need to minimize directly the Freidlin–Wentzell action functional and available algorithms include the minimum action method (MAM) [3], the adaptive MAM [14], the geometric MAM [8] and a high-order MAM [17].

Although it has been demonstrated that the minimum action method can be very useful for the study of noise-induced transitions and the structure of the phase space [3,15,16], the minimum action method and its variants are computationally demanding especially for spatially extended dynamical systems. This has limited the application of the method. The difficulty can be intuitively understood from the fundamental difference between deterministic and stochastic dynamics. To solve a deterministic dynamical system, we start from one point in the phase space and follow the trajectory given exactly by the equation. To study the transition between two states, we need to explore a set of transition paths, where due to the noise, any curve connecting the two states in the phase space can be a transition path. More specifically, the numerical difficulties include: (1) The Euler–Lagrange equation of the Freidlin–Wentzell (F–W) action functional corresponds to a *high-order nonlinear* $(d+1)$*-dimensional* boundary value problem, where the differentiation order with respect to both space and time will be doubled compared to the original partial differential equations (PDEs). Here d is the number of physical dimensions and the extra dimension is from the time direction. (2) After the discretization, the F–W action functional becomes a highly nonlinear function. A general and efficient way to compute the gradient is required by most efficient optimization solvers. (3) To accelerate the convergence of the optimization solver, an efficient preconditioner is desired, where a general one is usually not available for a nonlinear objective function and the efficiency of the preconditioner should be related to the dynamical system. (4) For large scale simulations, parallel computing is desired to enhance the numerical efficiency, where scalable algorithms are needed, especially for petascale or even exascale computing based on a heterogeneous high performance computing (HPC) architecture. However, the nonlinear differential operator in the physical space may not be in favor of direct parallelization. We now look at a real example. One classical dynamical system problem in fluid mechanics is the nonlinear instability of parallel shear flows, where the transition mechanism from a laminar state to a turbulence state is still not clear. This problem can be formulated and treated as a rare event [18], where the minimum action method is required since Navier–Stokes equations correspond to a non-gradient system. We here only emphasize the necessity of parallel computing in terms of the scale of the problem. The $d+1$-dimensional boundary value problem mentioned in issue (1) corresponds to a computation domain $[0,T]\times D$ with D being the physical domain. Roughly speaking, the time T should be of the same order as the Reynolds number. For example, for two-dimensional Poiseuille flows in a short channel, we are typically interested in Reynolds number located in the range (2900, 5772). Thus T can be very large. Although the deterministic Navier–Stokes equation may be simulated in physical domain D by a serial code, we need to consider the minimum action method to study the stochastic dynamics and it is impossible to deal with a computation domain, e.g., $[0, 8000]\times D$, using serial simulations even D is two-dimensional.

In this work, we mainly address the parallelization of the minimum action method in terms of the version proposed in [17]. We refer to this version as *hp*MAM for short, because it is formulated in the framework of *hp* finite element method, which allows both *h*-, *p*-, and *hp*-refinement for the transition path and can be easily coupled with finite element or spectral discretizations in the physical space. In *hp*MAM, components of the gradient of the F–W action functional are expressed explicitly as inner products, which can be efficiently computed by Gauss-type quadrature rules. In this paper, we will identify the essential parallelism of *hp*MAM. In particular, we present a simple but effective hybrid MPI/OpenMP parallelization strategy, which is consistent with the current high performance computing (HPC) architecture, e.g., a multi-socket multi-core symmetric multiprocessing cluster.

The paper is organized as follows. In Section 2, we describe the problem and briefly overview the high-order minimum action method. In Section 3, we identify the parallelism of our algorithm and present a hybrid MPI/OpenMP parallelization strategy. We test the scalability of the parallel *hp*MAM in Section 4, followed by a summary section.

2. Minimum action method

2.1. Description of the problem

We start from small random perturbations of an ordinary differential equation. Let $X_t = X(t):\mathbb{R}_+\to\mathbb{R}^n$ be a random process defined by the following stochastic ordinary differential equation (SODE):

$$dX_t = b(X_t)dt + \sqrt{\varepsilon}dW_t, \tag{1}$$

where W_t is a standard Wiener process in $\mathbb{R}^n$ and ε is a small positive parameter. Let $\boldsymbol{\phi}(t)\in\mathbb{R}^n$ be an absolutely continuous function defined on $t\in[0,T]$. The Freidlin–Wentzell theory [6] tells us that the probability of $X(t)$ passing through the δ-tube about $\boldsymbol{\phi}$ on $[0,T]$ is

$$\Pr(\rho(X,\boldsymbol{\phi})<\delta)\approx\exp\left(-\frac{1}{\varepsilon}S_T(\phi)\right), \tag{2}$$

where $\rho(\boldsymbol{\phi},\boldsymbol{\varphi})=\sup_{t\in[0,T]}|\boldsymbol{\phi}(t)-\boldsymbol{\varphi}(t)|$, and $S_T(\boldsymbol{\phi})$ is the action functional of $\boldsymbol{\phi}$ on $[0,T]$, defined as

$$S_T(\boldsymbol{\phi}) = \frac{1}{2}\int_0^T |\dot{\boldsymbol{\phi}} - b(\boldsymbol{\phi})|^2 dt = \frac{1}{2}\langle \dot{\boldsymbol{\phi}} - b(\boldsymbol{\phi}), \dot{\boldsymbol{\phi}} - b(\boldsymbol{\phi})\rangle_t, \tag{3}$$

where $\langle \mathbf{v}_1, \mathbf{v}_2\rangle_t$ indicates the inner product of vectors $\mathbf{v}_1(t), \mathbf{v}_2(t) \in \mathbb{R}^n$ on time interval $[0, T]$. In general, we have the following large deviation principle

$$\lim_{\varepsilon \longrightarrow 0} \varepsilon \log \Pr(X \in A) = -\min_{\boldsymbol{\phi}(t)\in A} S_T(\boldsymbol{\phi}), \tag{4}$$

where A is a particular set of random events. Hence, in analogy with the Laplace's method, the basic contribution to $\Pr(X \in A)$ is given by the neighborhood of the minimum of $S_T(\boldsymbol{\phi})$ when ε is small enough. The minimizer $\boldsymbol{\phi}^*$, which satisfies

$$S_T(\boldsymbol{\phi}^*) = \min_{\boldsymbol{\phi}\in A} S_T(\boldsymbol{\phi}) \tag{5}$$

is called the "minimal action path" (MAP). Depending on the definition of A, the MAP can capture many important phenomena induced by the small random perturbations, which can be rare but profound. Typical examples include chemical reactions, bistable genetic toggle switch, nucleation events during phase transitions, regime changes in climate, etc., from physical, biological and engineering applications.

For small random perturbations of a partial differential equation (SPDE)

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} = \mathcal{G}u(\mathbf{x}, t) + \sqrt{\varepsilon}\dot{W}(\mathbf{x}, t), \tag{6}$$

where $\mathbf{x} \in \mathbb{R}^d$, $d = 1, 2, 3$ is the physical dimension, $\mathcal{G}$ indicates a differential operator in the physical space and $\dot{W}(\mathbf{x}, t)$ is space–time white noise, the action functional is, in general, defined as [5]

$$S_T(u) = \frac{1}{2}\int_0^T \int_D (\partial_t u - \mathcal{G}u(\mathbf{x}, t))^2 d\mathbf{x}dt = \frac{1}{2}\langle \partial_t u - \mathcal{G}u, \partial_t u - \mathcal{G}u\rangle_{\mathbf{x},t}, \tag{7}$$

where $D \subset \mathbb{R}^d$ is the physical domain, $\langle\cdot,\cdot\rangle_{\mathbf{x},t}$ indicates the inner product with respect to both $\mathbf{x}$ and t. The minimal action path u^* for the action functional (7) is defined as

$$S_T(u^*) = \min_{u(\mathbf{x},t)\in B} S_T(u), \tag{8}$$

where B is the set of random events we are interested in about the stochastic dynamical system (6).

In this work, we focus on the parallelization strategy for numerical solutions of problems (5) and (8). For simplicity and without loss of generality, we consider problem (5) with the set A defined as

$$A = \{\boldsymbol{\phi}(0) = \mathbf{a}_1,\ \boldsymbol{\phi}(T) = \mathbf{a}_2\}. \tag{9}$$

where $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{R}^n$ are two points in the phase space. Then the MAP $\boldsymbol{\phi}^*$ is the most probable transition path from $\mathbf{a}_1$ to $\mathbf{a}_2$ induced by the small random perturbations. All our discussions later on about problem (5) can be generalized straightforwardly to deal with problem (8).

2.2. An adaptive high-order MAM (hpMAM)

We briefly overview the *hp*MAM proposed in [17]. We consider a high-order finite element approximation of $\boldsymbol{\phi}(t)$ with respect to a partition $\mathcal{T}_h$ of the time interval $[0, T]$:

$$\mathcal{T}_h: \quad 0 = t_0 < t_1 < \ldots < t_{N_e+1} = T,$$

where N_e is the number of finite elements. On a reference element $R = [-1, 1]$, we define $\mathcal{P}_p(R)$ as the space spanned by the following basis functions [12]:

$$\hat{\psi}_j(\tau) = \begin{cases} \frac{1-\tau}{2} & j = 0, \\ \frac{1-\tau}{2}\frac{1+\tau}{2}P_{j-1}^{1,1}(\tau), & 0 < j < p, \\ \frac{1+\tau}{2} & j = p, \end{cases} \tag{10}$$

where $P_j^{1,1}$ denote orthogonal Jacobi polynomials of degree j with respect to the weight function $(1-\tau)(1+\tau)$. $\hat{\psi}_0(\tau)$, and $\hat{\psi}_p(\tau)$ are consistent with the linear finite element basis, and $\hat{\psi}_i(\tau)$, $0 < j < p$, are introduced for high-order approximation. Note that $\hat{\psi}_j(\pm 1) = 0$ for $0 < i < p$. We call $\hat{\psi}_0(\tau)$ and $\hat{\psi}_p(\tau)$ left and right boundary modes, respectively, and $\hat{\psi}_j(\tau)$, $0 < j < p$, interior modes.

We then define the finite element approximation space for $\boldsymbol{\phi}(t)$ as

$$V_h^K = \{v : v \circ F_k^{-1} \in \mathcal{P}_p(R)\},$$
$$V_h = \{v \in H_0^1([0, T]) : v|_K \in V_h^K,\ K \in \mathcal{T}_h\},$$

where the function F_k defines an affine mapping from element $K = [t_k, t_{k+1}]$, $k = 0, 1, \ldots, N_e$, to the reference element $R = [-1, 1]$, and $v|_K$ is the restriction of v on element K. More specifically, V_h^K is spanned by the following basis functions

$$\psi_{k,j}(t) = \hat{\psi}_j(F_k(t)),\ t \in K = [t_k, t_{k+1}],\ k = 0, \ldots, N_e,\ j = 0, \ldots, p. \tag{11}$$

Thus V_h consists of piecewise polynomials up to order p. Note here that $V_h \subset H_0^1([0,T])$ due to the facts that $\boldsymbol{\phi}(0)$ and $\boldsymbol{\phi}(T)$ are specified and the action functional $S_T(\boldsymbol{\phi})$ takes the form of an inner product.

2.2.1. Gradient of the action functional

Let $i(k,j)$ indicate a global index for the basis functions $\psi_{k,j}(t) \in V_h$. Then $\boldsymbol{\phi}(t)$ has an approximation as

$$\boldsymbol{\phi}(t) \approx \boldsymbol{\phi}_h(t) = \sum_{i=1}^{M_t} \boldsymbol{\phi}_i \psi_{i(k,j)}(t), \quad \boldsymbol{\phi}_i \in \mathbb{R}^n, \tag{12}$$

where M_t is the total number of degrees of freedom. Correspondingly, problem (5) takes a discrete form as

$$S_T(\boldsymbol{\phi}_h^*) = S_T(\boldsymbol{\phi}_i^*|_{i=1,\ldots,M_t}) = \min_{\boldsymbol{\phi}_i \in \mathbb{R}^n, i=1,\ldots,M_t} S_T(\boldsymbol{\phi}_h), \tag{13}$$

which is an unconstrained optimization problem. It is well known that the gradient of the objective function is crucial for the efficiency of an optimization algorithm. We now derive the gradient of the action functional, i.e., $\partial S_T / \partial \phi_{i,l}$, where $\phi_{i,l}$ is the lth component of $\boldsymbol{\phi}_i$.

Let $\hat{b}(\boldsymbol{\phi})$ indicate the perturbation operator defined as

$$b(\boldsymbol{\phi}_h + \delta\boldsymbol{\phi}_h) \approx b(\boldsymbol{\phi}_h) + \hat{b}(\boldsymbol{\phi}_h)\delta\boldsymbol{\phi}_h, \tag{14}$$

where $\delta\boldsymbol{\phi}_h(t)$ is a perturbation function with $\delta\boldsymbol{\phi}_h(0) = \delta\boldsymbol{\phi}_h(T) = 0$. We then have the variation of the action functional as

$$\delta S_T(\boldsymbol{\phi}_h) = S_T(\boldsymbol{\phi} + \delta\boldsymbol{\phi}_h) - S_T(\boldsymbol{\phi}_h) = \langle \dot{\boldsymbol{\phi}}_h - b(\boldsymbol{\phi}_h), \delta\dot{\boldsymbol{\phi}}_h - \hat{b}(\boldsymbol{\phi}_h)\delta\boldsymbol{\phi}_h \rangle_t. \tag{15}$$

Note that $\delta\boldsymbol{\phi}_h$ also has an expansion in V_h

$$\delta\boldsymbol{\phi}_h(t) = \sum_{i=1}^{M_t} \delta\boldsymbol{\phi}_i \psi_i(t). \tag{16}$$

Thus

$$\delta S_T(\boldsymbol{\phi}_h) = \sum_{i=1}^{M_t} \langle \dot{\boldsymbol{\phi}}_h - b(\boldsymbol{\phi}_h), \delta\boldsymbol{\phi}_i\dot{\psi}_i - \hat{b}(\boldsymbol{\phi}_h)\delta\boldsymbol{\phi}_i\psi_i \rangle_t. \tag{17}$$

Consider the particular choice of $\delta\boldsymbol{\phi}_h$, whose coefficients are equal to zero except the lth component $\delta\phi_{i,l}$ of $\delta\boldsymbol{\phi}_i$. Eq. (17) becomes

$$\delta S_T(\boldsymbol{\phi}) = \langle \dot{\boldsymbol{\phi}}_h - b(\boldsymbol{\phi}_h), \dot{\psi}_i(t)\mathbf{e}_l - \hat{b}(\boldsymbol{\phi}_h)\psi_i(t)\mathbf{e}_l \rangle_t \delta\phi_{i,j},$$

which implies that

$$\frac{\partial S_T}{\partial \phi_{i,l}} = \langle \dot{\boldsymbol{\phi}}_h - b(\boldsymbol{\phi}_h), \dot{\psi}_i\mathbf{e}_l - \hat{b}(\boldsymbol{\phi}_h)\psi_i\mathbf{e}_l \rangle_t, \tag{18}$$

where $\mathbf{e}_l \in \mathbb{R}^n$ is the unit Euclidean vector such that its l-th component is 1 and the rest components are zero.

2.2.2. Time mesh adjustment

One difficulty of approximating the MAP is that the dynamics can significantly affect the quality of temporal discretization. Since we are looking for a curve in the phase space, we can also describe it by the arc length, i.e., the temporal discretization corresponds to an arc length discretization of the MAP. Specifically, the time element $[t_k, t_{k+1}]$ corresponds to the arc length element $\left[\int_0^{t_k} \sqrt{|\dot{\boldsymbol{\phi}}|^2}dt, \int_0^{t_{k+1}} \sqrt{|\dot{\boldsymbol{\phi}}|^2}dt\right]$. However, due to the nonlinear relation between time and arc length, a uniform discretization with respect to time may correspond to a highly nonuniform discretization with respect to arc length. For example, the time element $[t_k, t_{k+1}]$ has an element size $t_{k+1} - t_k$ while the corresponding arc length element has an element size $\int_{t_k}^{t_{k+1}} \sqrt{|\dot{\boldsymbol{\phi}}|^2}dt$, which is determined by $\dot{\boldsymbol{\phi}}$. In the transition region close to fixed points, the dynamics will become very slow, i.e., $\dot{\boldsymbol{\phi}}$ is close to zero, the arc length elements become very small and do not contribute to the approximation of the MAP. To improve the accuracy, we employ the moving mesh technique proposed in [14].

Let $s \in [0,1]$ indicate a scaled arc length such that the total length of the MAP is equal to 1. We need to find a mapping from a temporal discretization to a (nearly) uniform discretization with respect to s. A variational approach was used in [14], which minimizes the following functional

$$E(s) = \int_0^T w^{-1}(t)\left(\frac{ds}{dt}\right)^2 dt, \tag{19}$$

where $w(t)$ is a monitor function chosen as $w(t) = \sqrt{1 + C|\dot{\phi}|^2}$ with C being a positive constant. Note when C goes to infinity, $w(t) \sim |\dot{\phi}|$. The Euler–Lagrange equation of the functional (19) is

$$\begin{cases} \frac{d}{dt}\left(w^{-1}(t)\frac{ds}{dt}\right) = 0, & t \in (0,T), \\ s(0) = 0, & s(T) = 1. \end{cases} \tag{20}$$

For mesh adjustment, we first map the current time mesh to a discretization of $[0,1]$ with respect to s by solving Eq. (20). Second, we map a uniform discretization of $[0,1]$ with respect to s to a discretization of $[0,T]$ by computing $t^{-1}(s)$. This will be our new time mesh. Third, we project the current path $\phi_h(t)$ onto the new time mesh.

2.2.3. Nonlinear conjugate gradient method

Once the gradient of the action functional is obtained, we use the nonlinear conjugate gradient (CG) method to solve the optimization problem to get the MAP ϕ_h^*. Let $\mathbf{\Phi} \in \mathbb{R}^{M_t n}$ be a global vector whose components are $\phi_{i,l}$. The nonlinear CG method can be summarized as

$$\begin{cases} \mathbf{\Phi}_{m+1} = \Phi_m + \alpha_m \mathbf{d}_m, \\ \mathbf{d}_{m+1} = -\mathbf{g}_{m+1} + \beta_m \mathbf{d}_m, \quad \mathbf{d}_0 = -\mathbf{g}_0, \end{cases} \tag{21}$$

where the subscript m indicates the iteration step, the positive step size α_m is obtained by a line search algorithm, $\mathbf{g}_m = \nabla S_T(\mathbf{\Phi}_m)$, and β_m is the CG update parameter. We define $\beta_m = \max\{\hat{\beta}_m, \eta_m\}$ as in [7]:

$$\hat{\beta}_m = \left(\mathbf{y}_m - 2\mathbf{d}_m \frac{|\mathbf{y}_m|^2}{\mathbf{d}_m^{\mathrm{T}}\mathbf{y}_m}\right)^{\mathrm{T}} \frac{\mathbf{g}_{m+1}}{\mathbf{d}_m^{\mathrm{T}}\mathbf{y}_m}, \quad \eta_k = \frac{-1}{|\mathbf{d}_m| \min\{0.01, |\mathbf{g}_m|\}} \tag{22}$$

with $\mathbf{y}_m = \mathbf{g}_{m+1} - \mathbf{g}_m$.

When the preconditioning is desired, we consider a new variable $\Phi = \mathbf{S}\hat{\Phi}$, where $\mathbf{S}$ is an invertible matrix chosen to speed-up the convergence. Writing the nonlinear CG method with respect to $\hat{\mathbf{\Phi}}$ and converting it back to $\mathbf{\Phi}$ we obtain the preconditioned nonlinear CG method:

$$\begin{cases} \Phi_{m+1} = \Phi_m + \alpha_k \mathbf{d}_m, \\ \mathbf{d}_{m+1} = -\mathbf{P}\mathbf{g}_{m+1} + \bar{\beta}_m \mathbf{d}_m, \quad \mathbf{d}_0 = -\mathbf{P}\mathbf{g}_0, \end{cases} \tag{23}$$

where $\mathbf{P} = \mathbf{S}\mathbf{S}^{\mathrm{T}}$. The parameter $\bar{\beta}_m$ is the same as β_m except that $\mathbf{g}_m$ and $\mathbf{d}_m$ are replaced by $\mathbf{S}^{\mathrm{T}}\mathbf{g}_m$ and $\mathbf{S}^{-1}\mathbf{d}_m$, respectively. However, we do not need to know $\mathbf{S}$ explicitly by observing that $(\mathbf{S}^{\mathrm{T}}\mathbf{g}_k)^{\mathrm{T}}(\mathbf{S}^{\mathrm{T}}\mathbf{g}_m) = \mathbf{g}_m^{\mathrm{T}}\mathbf{S}\mathbf{S}^{\mathrm{T}}\mathbf{g}_m = \mathbf{g}_k^{\mathrm{T}}\mathbf{P}\mathbf{g}_m$ and $\left(\mathbf{S}^{-1}\mathbf{d}_k\right)^{\mathrm{T}}(\mathbf{S}^{\mathrm{T}}\mathbf{y}_m) = \mathbf{d}_m^{\mathrm{T}}\mathbf{S}^{-\mathrm{T}}\mathbf{S}^{\mathrm{T}}\mathbf{y}_m = \mathbf{d}_m^{\mathrm{T}}\mathbf{y}_m$. Thus we only need to know the matrix $\mathbf{P}$. An effective preconditioner $\mathbf{P}$ is, in general, an approximation of the Hessian. Unfortunately, it is usually difficult to find an effective preconditioner for a general nonlinear function, which is heavily problem dependent. In [17], it was shown that the inverse of the linear part of the Euler–Lagrange equation of the F–W action functional may serve as a good preconditioner.

Algorithm 1. Adaptive high-order MAM

Project the initial path $\phi(t)$ onto a uniform time mesh $\mathcal{T}_h$ of $[0,T]$ and define $\mathbf{\Phi}_0$ which is a global vector containing all unknown coefficients of the finite element approximation;

Start the iteration of nonlinear CG solver (21)

$$\mathbf{\Phi}_{m+1} = \mathbf{\Phi}_m + \alpha_m \mathbf{d}_m$$

Check the mesh quality very m iteration steps.

- Compute the arc length for each element according to the monitor function $w(t)$.
- If the ratio r_s between the largest arc length and the smallest one is larger than a prescribed threshold, solve equation (20) to obtain a new time mesh.
- Project the current path onto the new time mesh and update $\mathbf{\Phi}_m$.

Stop the CG iteration when error tolerance or the maximum iteration number is achieved.

3. Hybrid Parallelization of *hp*MAM

In this section we identify the main parallelism of Algorithm 1. Algorithm 1 mainly consists of the following four components: evaluation of $S_T(\phi_h)$, evaluation of the gradient $\nabla S_T(\phi_h)$, time mesh adjustment and nonlinear CG iteration, for which we consider parallel implementations.

We note that parallelizing the evaluations of $S_T(\boldsymbol{\phi}_h)$ and $\nabla S_T(\boldsymbol{\phi}_h)$ should depend on the structure of $b(\boldsymbol{\phi}_h)$, which determines the perturbation operator $\dot{b}(\boldsymbol{\phi}_h)$. Since $b(\boldsymbol{\phi}_h)$ is usually a nonlinear function of $\boldsymbol{\phi}_h$, which is not in favor of a direct parallelization, we will assume that no explicit information is known about $b(\phi_h)$ and will not incorporate its structure into the parallelization either, although we believe that such an incorporation may generate a more efficient parallelization strategy, especially for a PDE problem.

3.1. Parallelizing the time direction

Due to the employment of the finite element framework, it is natural to first consider a decomposition of the partition $\mathcal{T}_h$. Let N_{node} denote the number of computation nodes. For simplicity, we assume that the polynomial order in each time element is the same and $r_1 = \frac{N_e+1}{N_{\text{node}}}$ is an integer. In other words, each computation node will be assigned r_1 elements and the loading balance is about uniformly distributed.

Let e_k denote the element $[t_k, t_{k+1}],\ k = 0, \ldots, N_e$. Let $\mathcal{T}_h^i,\ i = 0, \ldots, N_{\text{node}} - 1$, indicate the sub-partition on computation node i, which consists of elements $e_{ir_1}, \ldots, e_{(i+1)r_1-1}$. Let $e_k^i = e_{ir_1+k},\ k = 0, 1, \ldots, r_1 - 1$, indicate the kth element in the sub-partition $\mathcal{T}_h^i$. Then $\boldsymbol{\phi}_h(t)$ can be written as

$$\boldsymbol{\phi}_h(t) = \sum_{i=0}^{N_{\text{node}}-1}\sum_{k=0}^{r_1-1}\sum_{j=0}^{p}\boldsymbol{\phi}_{k,j}^i\psi_{k,j}^i(t), \quad \boldsymbol{\phi}_{k,j}^i \in \mathbb{R}^n. \tag{24}$$

where $\psi_{k,j}^i(t)$ is the jth basis function in element e_k^i located in sub-partition $\mathcal{T}_h^i$ (see Eq. (11)). Due to the continuity, the coefficient of the right boundary mode in e_k should be equal to the coefficient of the left boundary mode in e_{k+1}, which implies that for two adjacent sub-partitions $\mathcal{T}_h^i$ and $\mathcal{T}_h^{i+1}$, the coefficients $\boldsymbol{\phi}_{r_1-1,p}^i$ and $\boldsymbol{\phi}_{0,0}^{i+1}$ should be the same. Since all interior modes are equal to zero on boundaries, $\boldsymbol{\phi}_{r_1-1,p}^i = \boldsymbol{\phi}_{0,0}^{i+1}$ is the only information shared by the i- and $(i+1)$th adjacent computation nodes. We call such a parallelization of time elements as level-one parallelization, which indicates the parallelization between computation nodes. The level-one parallelization can be achieved efficiently by message passing interface (MPI), see Section 3.4, where each MPI process is associated with one computation node and deals with a certain number of time elements.

Although such a basic parallelization strategy has a good scalability, we immediately have a problem that the maximum number of computation nodes is limited by the number of elements. Such a problem becomes more severe when a high order p is employed or the dimension of $\boldsymbol{\phi}_h(t)$, i.e., n, is large. One typical example is small random perturbations of partial differential equations, where n can be arbitrarily large. One possible solution is that we can also parallelize the components of $\boldsymbol{\phi}_h(t)$, which corresponds to a domain decomposition in the physical space if a partial differential equation is considered. However, for such a parallelization strategy the most favorable case is that equations for the components of $\boldsymbol{\phi}_h(t)$ are decoupled, which rarely occurs in practice. Otherwise, the communication scenario between computation nodes is similar with an N-body problem, which is not scalable at all unless $b(\boldsymbol{\phi}_h)$ has some special properties we can use. In other words, parallelizing $\boldsymbol{\phi}_h(t)$ is, in general, not an efficient strategy for level-one parallelization without any acknowledgement of the structure of $b(\boldsymbol{\phi}_h)$.

3.2. Parallelizing the gradient

We first note that both $S_T(\boldsymbol{\phi}_h)$ and $\nabla S_T(\boldsymbol{\phi}_h)$ have an integration form, which corresponds to an element-wise expression

$$S_T(\boldsymbol{\phi}_h) = \sum_{i=0}^{N_{\text{node}}-1}\sum_{k=0}^{r_1-1}\frac{1}{2}\left\langle \dot{\boldsymbol{\phi}}_h|_{e_k^i} - b\left(\boldsymbol{\phi}_h|_{e_k^i}\right), \dot{\boldsymbol{\phi}}_h|_{e_k^i} - b\left(\boldsymbol{\phi}_h|_{e_k^i}\right)\right\rangle_t \tag{25}$$

$$\frac{\partial S_T}{\partial \boldsymbol{\phi}_{k,j,l}^i} = \left\langle \dot{\boldsymbol{\phi}}_h|_{e_k^i} - b\left(\boldsymbol{\phi}_h|_{e_k^i}\right), \dot{\psi}_{k,j}^i\mathbf{e}_l - \hat{b}\left(\boldsymbol{\phi}_h|_{e_k^i}\right)\psi_{k,j}^i\mathbf{e}_l\right\rangle_t, \tag{26}$$

where $\phi_{k,j,l}^i$ is the lth component of the coefficient associated with the jth basis function on element e_k^i of sub-partition $\mathcal{T}_h^i$. Depending on the polynomial order of $\boldsymbol{\phi}_h(t)$ and the nonlinearity of $b(\boldsymbol{\phi}_h)$ and $\hat{b}(\boldsymbol{\phi}_h)$, the element-wise inner product can be well approximated using a sufficient number of Gauss-type quadrature points.

Since we do not take into account the structure of $b(\boldsymbol{\phi}_h)$, the parallelization for $S_T(\boldsymbol{\phi}_h)$ cannot go further than an element-wise strategy when the level-one parallelization is considered. However, the situation for $\nabla S_T(\boldsymbol{\phi}_h)$ is different. First, when the dimension n is large, the computation of $\nabla S_T(\boldsymbol{\phi}_h)$ can be much more expensive than the computation of $S_T(\boldsymbol{\phi}_h)$. Parallelization is more desired for the gradient. Second, each component of $S_T(\boldsymbol{\phi}_h)$ is associated with one basis function in element e_k^i through an element-wise inner product. All these inner products share the element-wise information about $\dot{\boldsymbol{\phi}}_h|_{e_k^i}$, $b(\boldsymbol{\phi}_h|_{e_k^i})$ and $\hat{b}(\boldsymbol{\phi}_h|_{e_k^i})$. Other than that, the computations of $\partial S_T/\partial\phi_{k,j,l}^i$ are actually independent of each other, which are determined only by the associated basis functions $\psi_{k,j}^i\mathbf{e}_l$. Such an independence gives us a chance to compute $\nabla S_T(\boldsymbol{\phi}_h)$ in parallel once the shared information is obtained. Obviously, such a parallelization with respect to the components of ∇S_T is not involved with communications between computation nodes. For such a reason, we call it level-two parallelization, which happens within each computation node. Level-two parallelization can be achieved by open multi-processing (OpenMP), see Section 3.4, where multiple threads are used and each thread deals with a certain number of components of the

gradient of the action functional. Apparently, level-two parallelization can also be achieved through GPU computing for more efficiency. In this work, we only consider OpenMP for simplicity.

Remark 1. Before computing the action functional and its gradient, the global vector $\mathbf{\Phi}$ used by the nonlinear CG solver will be scattered to each time element. Once the element-wise operations are finished, all information will be gathered and passed to the nonlinear CG solver. The scattering and gathering operations are also parallelized through OpenMP using a mapping between the global vector and the local vectors in each time element.

3.3. Parallelization of the time mesh adjustment

For the time mesh adjustment, two things need to be done: compute a new time mesh and project the current MAP to the new time mesh. To obtain a new time mesh, we use a second-order finite element method to solve Eq. (20), since we only need a nearly uniform mesh with respect to arc length and the inverse $t^{-1}(s)$ can be computed explicitly for second-order piecewise polynomials. Furthermore, solving Eq. (20) corresponds to inverse a tridiagonal stiffness matrix, whose cost is just linear of $O(N_e)$. The projection from the old time mesh to the new one will be done element-wisely, corresponding to a cost $O(p^2N_e)$, which is more expensive than obtaining a new time mesh. Actually, it could be not necessary to consider a parallel solver of Eq. (20) due to the linear cost. We can use one computation node to compute and broadcast the new time mesh. The parallelization for the projection operation can be a little complicated due to the fact that we keep a fixed number of elements in each computation node instead of a fixed time interval. Determined by the skewness of the old mesh, one computation node may need to talk to quite a number of computation nodes for the information of $\boldsymbol{\phi}_h(t)$, which implies that the loading on each computation node and communications between computation nodes can be highly nonuniform. Fortunately, such a situation will mainly happen in the early stage of the nonlinear CG iteration and will be improved as the iteration converges. Due to the fact that the mesh only needs to be checked every a certain number of iteration steps and the frequency for mesh adjustment will decay as the mesh is being stabilized, the possible low efficiency for the parallelization of the projection operation is overall not important.

3.3.1. Parallelizing the projection

First of all, the projection will be implemented element-wisely. Second, due to the reparametrization the projection onto a certain element may need to communicate with more than one sub-partitions of the old time mesh. Based on these two facts, we parallelize the projection as follows:

- We keep a copy of global time mesh on each computation node although each computation node only deals with a sub-partition.
- On the ith computation node, we classify the elements in the new sub-partition $\hat{\mathcal{T}}_h^i$ into two groups: G_1^i and G_2^i. G_1^i includes the elements for which the projection can be determined by the old sub-partition $\mathcal{T}_h^i$; G_2^i includes the elements for which the projection needs information from other sub-partitions.
- For elements in G_1^i, the projection can be done locally without communications between computation nodes; For elements in G_2^i, we need all computation nodes to work together in a proper way to exchange information appropriately. Since we keep a copy of the global time mesh on each computation node, a unique rule can be generated simultaneously and followed by all computation nodes to deal with elements in G_2^i in parallel.

As an illustration, an example can be found in Fig. 1, where both the old time mesh and the new one have four three-element sub-partitions. It can be seen that the first element $\hat{e}_{1,0}$ of $\hat{\mathcal{T}}_h^1$ needs information from $\mathcal{T}_h^0$ and $\mathcal{T}_h^1$, and the first element $\hat{e}_{3,0}$ of $\hat{\mathcal{T}}_h^3$ needs information of $\mathcal{T}_h^2$ and $\mathcal{T}_h^3$. The projections for elements $\hat{e}_{1,0}$ and $\hat{e}_{3,0}$ can be implemented in parallel. We also note that projections for all elements in $\hat{\mathcal{T}}_h^0$, the second element in $\hat{\mathcal{T}}_h^1$, all elements in $\hat{\mathcal{T}}_h^2$ and the last two elements in $\hat{\mathcal{T}}_h^3$ can be implemented in parallel since they only require information from the corresponding old sub-partitions, which is available locally on each computation node.

We note that the parallelization of the projection can be a mix of level-one and level-two parallelization. All aforementioned parallelization of the projection can be implemented through MPI since the parallelization is with respect to the sub-partition $\hat{\mathcal{T}}_h$, i.e., computation node or MPI process. Within each computation node, it is possible to implement level-two

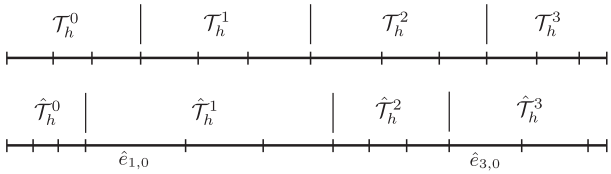

Fig. 1. Illustration of the parallelization of the projection from the old time mesh to the new one.

parallelization. For example, the three elements in $\hat{\mathcal{T}}_h^0$ only depend on the local information about $\mathcal{T}_h^0$ and the projection can be implemented by three threads in parallel. However, some elements in $\hat{\mathcal{T}}_h^0$ may share a certain element in $\mathcal{T}_h^0$. If we associate one thread to each of those elements in $\hat{\mathcal{T}}_h^0$, we need to synchronize those threads to make sure that the memory of the shared element in $\mathcal{T}_h^0$ is used correctly without being updated simultaneously by more than one thread. Considering the linear cost of the overall projection, we do not consider the level-two parallelization for the projection, i.e., the projection of the three elements in $\hat{\mathcal{T}}_h^0$ will be implemented element-by-element serially on computation node 0.

3.4. A hybrid MPI/OpenMP parallel strategy

To obtain an efficiently scalable parallel algorithm, we need to take into account the underlying hardware architecture. The mainstream of current high performance computing (HPC) architecture is designed to be highly hierarchical. A typical multi-socket multi-core symmetric multiprocessing (SMP) cluster is shown in Fig. 2, where SMP nodes are coupled via high-speed interconnect network. Inside each SMP nodes, two or more multi-core processors are connected to a single shared main memory, where each core can work independently as a single processor.

There exist two typical methods: message passing interface (MPI) and open multi-processing (OpenMP) to achieve parallelism in a program. Both methods can be applied to the aforementioned hybrid clusters. However, we need to be aware of the difference between these two methods to achieve parallelism. MPI assumes that all parallel processes have their own memory. All MPI processes communicate with each other by sending and receiving messages, no matter that they are located on the same SMP node or not. In other words, an MPI process will always copy the received message to its own memory. However, OpenMP assumes that all its threads share the same main memory. When communications occur, one thread only need to tell others where the information is located in the main memory and no copy is implemented. Thus it is natural to expect that OpenMP can be more efficient on a single SMP node, although it is possible to extend OpenMP to the whole SMP cluster using "distributed virtual shared memory" technology like Intel Cluster OpenMP.

Although there is evidence that a pure MPI program can be more efficient, where every CPU core is treated as a separate MPI entry with its own address space, a pure MPI strategy is not suitable here because we assume no direct parallelization, or level-one parallelization, of the components of $b(\boldsymbol{\phi}_h)$. We instead focus on a hybrid MPI/OpenMP strategy, which uses OpenMP for parallelization inside a SMP node and MPI for message passing between SMP nodes. More specifically, we use MPI for level-one parallelization, which indicates the parallelization of time elements between SMP nodes, and OpenMP for level-two parallelization, which indicates the parallelization within SMP nodes. For each SMP node, we assign one MPI process, which deals with r_1 elements corresponding to a certain sub-partition $\mathcal{T}_h^i$. Any two adjacent MPI processes only share the information $\boldsymbol{\phi}_{r_1-1,p}^i = \boldsymbol{\phi}_{0,0}^{i+1}$, which is the minimum amount of information if the components of $\boldsymbol{\phi}_h$ are not parallelized directly. For level two parallelization, the shared information needs to be obtained first before the parallelization can be implemented. More specifically, after we obtain the shared information $\dot{\boldsymbol{\phi}}_h|_{e_k}$, $b(\boldsymbol{\phi}_h|_{e_k})$ and $\hat{b}(\boldsymbol{\phi}_h|_{e_k})$, we can distribute the components $\partial S_T/\partial\phi_{k,j,l}^i$ located in sub-partition $\mathcal{T}_h^i$ uniformly to OpenMP threads for independent computations, where $b(\boldsymbol{\phi}_h|_{e_k})$ is located in the single main memory and can be used by any thread whenever necessary.

Although a hybrid MPI/OpenMP parallel strategy seems more suitable, it is not trivial to generate an efficient hybrid parallel minimum action method. For example, there does not exist a standard criterion to choose the number of MPI processes and the number of OpenMP threads. In other words, there do not exist a simple answer to the fundamental question of how to map the algorithm to the hybrid HPC architecture to achieve the best scalability. We will study this issue through numerical experiments.

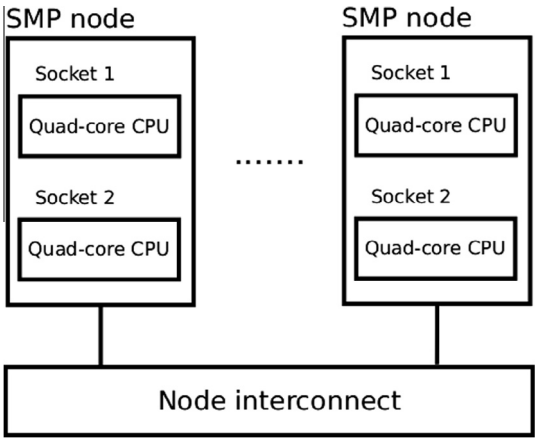

Fig. 2. A typical multi-socket multi-core SMP cluster.

Algorithm 2. Parallel adaptive high-order MAM

Decompose the time partition $\mathcal{T}_h$ uniformly with respect to available SMP nodes. Each SMP node is associated with one MPI process, and deals with a sub-partition $\mathcal{T}_h^i$, $i = 0, \ldots, N_{\text{node}} - 1$.

Project the initial path $\phi(t)$ onto sub-partitions $\mathcal{T}_h^i$ of $[0, T]$ and define $\mathbf{\Phi}_0^i$ which is a vector containing all unknown coefficients related to $\mathcal{T}_h^i$.

Start the iteration of nonlinear CG solver (21)

$$\mathbf{\Phi}_{m+1}^i = \mathbf{\Phi}_m^i + \alpha_m \mathbf{d}_m^i,$$

where the gradient of the action functional and the vector operations are computed in parallel through both MPI and OpenMP.

Check the mesh quality every m iteration steps.

- Compute the arc length for each element according to the monitor function $w(t)$.
- Compute r_s^i for each sub-partition $\mathcal{T}_h^i$, which is the ratio between the largest arc length and the smallest one. If $\max_{i=0,\ldots,N_{\text{node}}-1} r_s^i$ is larger than a prescribed threshold, solve Eq. (20) to obtain a new time mesh.
- Project the current path onto the new time mesh and update $\mathbf{\Phi}_m^i$, where the projection is implemented in parallel though MPI.

Stop the CG iteration when error tolerance or the maximum iteration number is achieved.

3.5. Comments on the parallel MAM for stochastic partial differential equations

The scenario of the MAM for partial differential equations is similar with the MAM for ordinary differential equations with a large n. For partial differential equations we need to discretize the physical space. Assume that the approximation space is spanned by $\{h_i(\mathbf{x})\}_{i=1}^{M_\mathbf{x}}$ such that $u(\mathbf{x}, t)$ can be approximated as

$$u(\mathbf{x}, t) \approx u_h(\mathbf{x}, t) = \sum_{i=1}^{M_\mathbf{x}} \sum_{j=1}^{M_t} u_{ij} h_i(\mathbf{x}) \psi_j(t).$$

Define the perturbation operator $\hat{\mathcal{G}}$ as

$$\mathcal{G}(u + \delta u) = \mathcal{G}u + \hat{\mathcal{G}}\delta u + O(\delta^2 u).$$

Then the action functional and its gradient take the form as [17]

$$S_T(u_h) = \frac{1}{2} \langle (\partial_t - \mathcal{G})u_h, (\partial_t - \mathcal{G})u_h \rangle_{\mathbf{x},t}, \tag{27}$$

$$\frac{\partial S_T(u_h)}{\partial u_{ij}} = \langle (\partial_t - \mathcal{G})u_h, (\partial_t - \hat{\mathcal{G}})(h_i(\mathbf{x})\psi_j(t)) \rangle_{\mathbf{x},t}. \tag{28}$$

First of all, the number $M_\mathbf{x}$ of degrees of freedom in the physical space can be relatively large, especially for two- and three-dimensional cases, parallelizing the gradient $\nabla S_T(u_h)$ for SPDEs is more desired than that for SODEs. Second, before computing $S_T(\phi_h)$ and $\nabla S_T(\phi_h)$ for SODEs, $b(\phi_h)$ and the perturbation operator $\hat{b}(\phi_h)$ will be computed on quadrature points. Since both $b(\phi_h)$ and $\hat{b}(\phi_h)$ are not a differential operator, they only need to be computed once. Then the element-wise cost for $S_T(\phi_h)$ is the cost for $b(\phi_h)$ plus the cost for the inner product. The element-wise cost for $\nabla S_T(\phi_h)$ is the cost for $n(p+1)$ inner products and the cost for $\hat{b}(\phi_h)$. Thus, the cost for each component of $\nabla S_T(\phi_h)$ is much cheaper than that for $S_T(\phi_h)$ on average. However, for SPDEs, $\hat{\mathcal{G}}h_i(\mathbf{x})$ needs to be computed for each $h_i(\mathbf{x})$ since $\hat{\mathcal{G}}$ is usually a differential operator in the physical space and depends on u_h, which implies that the cost for $S_T(u_h)$ is comparable with that for each component of $\nabla S_T(u_h)$. In this sense, the level-two parallelization is more important for SPDEs.

4. Numerical examples

For numerical study, we focus on the following two cases:

(i) The number of time elements is much larger than the dimensionality, i.e., $N_e \gg n$. For this case we expect that the hybrid strategy should be biased to MPI for more efficiency.
(ii) The number of time elements is of the same order as the dimensionality, i.e., $N_e \sim n$. For this case it is not clear about the best balance between MPI and OpenMP and we need to clarify it through numerical experiments.

We will use both SODE and SPDE examples to discuss the above two cases. In particular, we use the SODE example to study case (i) and the SPDE example to study cases (ii).

4.1. Study of case (i): $N_e \gg n$

For this case, we consider the following example, for which the MAP can be obtained explicitly:

$$\begin{cases} dx = -\partial_x V(x,y)dt + \sqrt{\varepsilon}dW_t^x \\ dy = -\partial_y V(x,y)dt + \sqrt{\varepsilon}dW_t^y \end{cases} \tag{29}$$

where the potential $V(x,y)$ is

$$V(x,y) = (1-x^2-y^2)^2 + y^2/(x^2+y^2). \tag{30}$$

The dynamical system has two stable fixed points $a_1 = (-1,0)$ and $a_2 = (1,0)$, which are local minima of the potential $V(x,y)$. We consider the MAP in the upper half-plane connecting a_1 and a_2 through the saddle point $a_3 = (0,1)$. Then the explicit form of this MAP is the upper branch of the unit circle: $x^2+y^2 = 1$. The exact action functional is $2 \times (V(a_3) - V(a_1)) = 2$.

In the following numerical experiments, the numbers of elements are just chosen for the scalability study, which can be much larger than the number of elements needed to identify the MAP between a_1 and a_2. In the left plot of Fig. 3 we plot the strong scaling for the problem with 1024000 linear elements for the discretization, where a pure MPI parallelization is considered and the solution time is taken as the mean solution time given by 10 independent runs. Since each computation node we used has 8 cores, we use the results given by the parallel computation with 8 MPI processes as the reference for the strong scaling. Super linear speedup is observed up to 1024 cores, which should be due to the cache effect resulting from the memory hierarchy of the computer. From the algorithm point of view, the main bottleneck for the strong scaling seems to be the cost for adaptive mesh adjustment. In the right plot of Fig. 3, we plot the relative cost of the mesh adjustment defined as the ratio between the time for mesh adjustment and the solution time. We then increase the polynomial order in each element from 1 to 4, and plot the strong scaling for this problem in the left plot of Fig. 4 and the relative cost of the mesh adjustment in the right plot of the same figure. Strong scaling is again observed until the relative cost of the mesh adjustment becomes important.

We subsequently test the strong scaling of OpenMP for a relatively small problem. The solution time for each case is the mean time of 10 independent runs. Here each thread corresponds to one core on a computation node. In the left plot of Fig. 5, we plot the strong scaling for a problem where 1024 linear elements are used for the discretization. The solid line indicates the Amdahl's law defined as

$$S = \frac{1}{\frac{f_{\text{par}}}{P_c} + (1 - f_{\text{par}})}, \tag{31}$$

where S is the parallel speedup, f_{par} the parallel fraction of the code and P_c the number of cores. The parallel fraction f_{par} is estimated using the solution time given by one- and two-core simulations, which is about 95%. It is seen that the speedup agrees well with the Amdahl's law up to 5 cores. In the right plot of Fig. 5, we plot the strong scaling for a problem where 10240 linear elements are used for the discretization. The solid line indicates the linear scaling. It is seen that a superlinear scaling is obtained for up to 8 cores. Note here for the OpenMP tests we switch off the mesh adaptivity since the time mesh

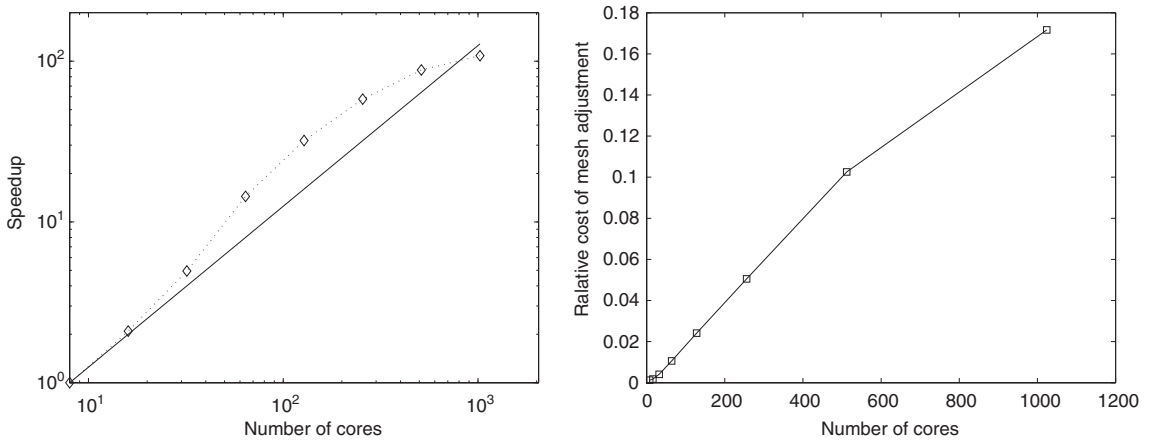

Fig. 3. 102400 linear elements are used for the discretization. Left: Strong scaling with respect to the solution time for a pure MPI parallelization. Right: Relative cost of mesh adjustment.

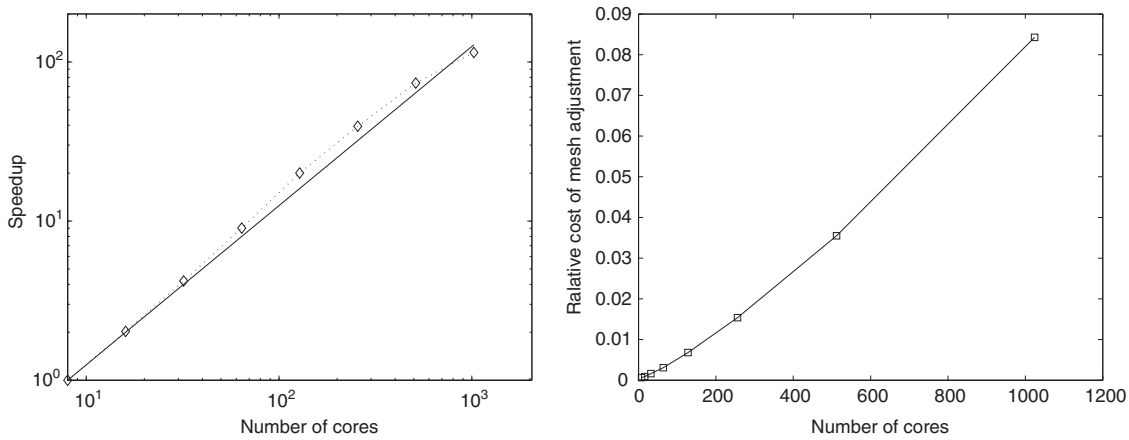

Fig. 4. 102400 elements with polynomial order 4 are used for the discretization. Left: Strong scaling with respect to the solution time for a pure MPI parallelization. Right: Relative cost of mesh adjustment.

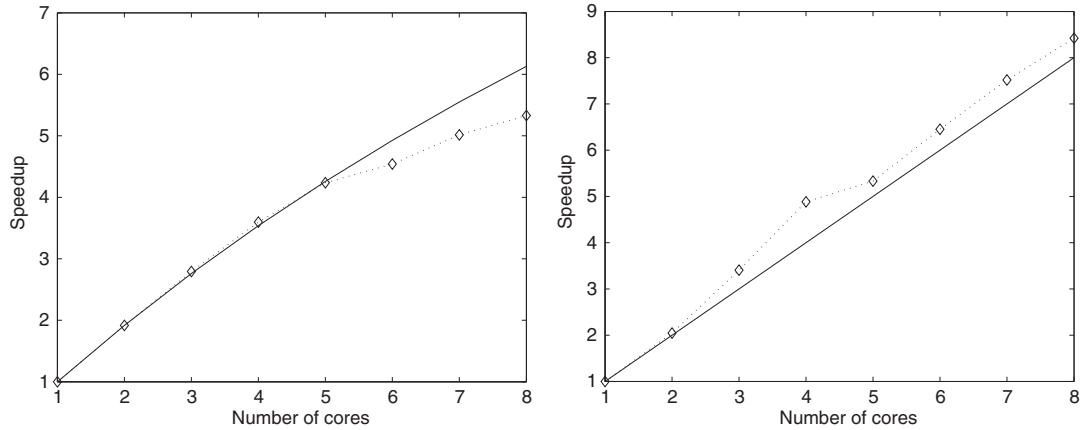

Fig. 5. Strong scaling of parallelization through OpenMP. Left: 1024 linear elements are used for the discretization. The solid line indicates the Amdahl's law, where the parallel fraction of the code is estimated using the results given by 2 threads. Right: 10240 linear elements are used, where the solid line indicates the linear scaling.

adjustment is parallelized through MPI. We observe that the efficiency of the parallelization through OpenMP varies according the problem size, where a superlinear speedup can be achieved for a certain range of problem sizes. Such a phenomena is also related to the memory hierarchy of the computer.

We now consider the strong scaling of the hybrid code, where the number of threads is fixed on each computation node. In Fig. 6 we plot the speedup with respect to the number of computation nodes. It is seen that linear or superlinear scaling is obtained when the number of nodes is relatively small ($\leqslant 32$). As the problem size becomes smaller on each node, the intra-node communications from OpenMP will deteriorate the strong scaling. We also observe that the strong scaling with 4 fixed threads on each node is better than that with 8 threads on each node. This is also consistent with our observations about the strong scaling of the pure MPI code, where the strong scaling is observed up to 1024 cores for the same problem size.

In summary, a good strong scaling has been observed for the pure MPI code. The scaling of the hybrid code also depends on the problem size assigned to each computation node. From the algorithm point of view, the main bottleneck for the strong scaling is from the time mesh adjustment. More specifically, it depends on how often the mesh adjustment is implemented and how soon the adaptive mesh becomes stabilized. Both of these two issues are problem dependent. Due to the robustness of the *hp*MAM [17], it is, in general, not necessary to check the mesh quality very often, which can weaken the effect of the adaptivity on the strong scaling. The second issue is mainly related to the complexity of the structure of the phase space. The simpler the structure of the phase space is, the sooner the adaptive time mesh becomes stabilized.

4.2. Study of case (ii): $N_e \sim n$

For this case, we consider the two-dimensional Navier–Stokes equations perturbed by small divergence-free space–time white noise, which can be regarded as a typical application of minimum action method to spatially extended non-gradient

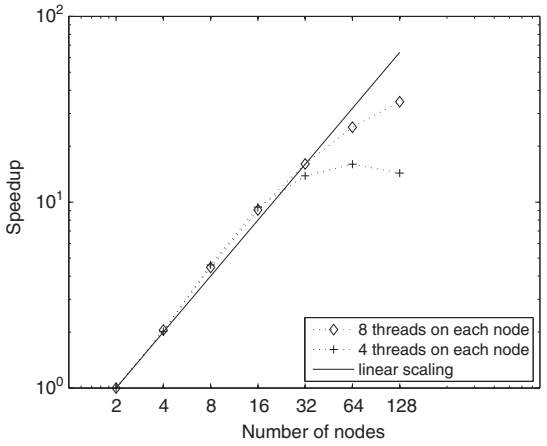

Fig. 6. Strong scaling of the hybrid code. 102400 linear elements are used for the discretization. The scaling is with respect to the number of nodes, where on each node the number of threads are fixed. The speedup is computed with respect to the solution time given by the two-node simulations.

systems. In particular, we use two-dimensional Poiseuille flow as an example. The mathematical model takes the following form:

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\Delta\mathbf{u} + \sqrt{\varepsilon}\dot{W}(\mathbf{x},t), \\ \nabla\cdot\mathbf{u} = 0, \end{cases} \tag{32}$$

where $\mathbf{u} = (u, v) \in \mathbb{R}^2$, Re is the Reynolds number, and $\dot{W}(\mathbf{x}, t)$ is divergence-free space–time white noise. We define the physical domain $(x, y) \in D := [0, 2\pi] \times [-h, h]$, where h is a positive real number. The boundary conditions are then given as

$$\begin{cases} \mathbf{u}|_{x=0} &= \mathbf{u}|_{x=2\pi} \\ p|_{x=0} &= p|_{x=2\pi} \\ \mathbf{u}|_{y=\pm h} &= 0 \end{cases} \tag{33}$$

i.e., we consider periodic boundary conditions in x direction and non-slip boundary condition at walls $y = \pm h$. The base flow takes the form

$$\mathbf{u}_b = (1 - (y/h)^2, 0), \quad p_b = -\frac{2}{Re}\frac{x}{h^2}. \tag{34}$$

One important problem in fluid mechanics is the instability of parallel shear flows. The linear stability theory or the Orr–Sommerfeld equation yields a critical number $Re_L \approx 5772$ for two-dimensional Poiseuille flows, beyond which the base flow is linearly unstable. The nonlinear stability theory says that there exists a global critical Reynolds number $Re_G \approx 2900$, beyond which another stable traveling wave solution can exist and below which the base flow is the only stable solution. Nonmodal stability theory or energy theory gives the critical Reynolds number $Re_E \approx 89$, below which the energy of any perturbation will decay monotonically. However, all these critical Reynolds numbers cannot describe the stochastic dynamics under noise, since the transition between the base flow and another stable solution is allowed. The minimum action method can help us understand this classical problem in the probabilistic sense, which is more general. More discussions about this problem and the numerical method can be found in [18].

The action functional for the stochastic Navier–Stokes Eq. (32) can be rewritten as

$$S_T(\mathbf{u}, p) = \frac{1}{2}\int_0^T \left\| \frac{\partial \mathbf{u}}{\partial t} - (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \frac{1}{Re}\Delta\mathbf{u} \right\|_2^2 dt, \tag{35}$$

where $\|\cdot\|$ indicates the L_2 norm in the physical space [18]. We then consider the optimization problem

$$S_T(\mathbf{u}^*, p^*) = \min_{(\mathbf{u},p)\in A} S_T(\mathbf{u}, p), \tag{36}$$

subject to the divergence free condition $\nabla\cdot\mathbf{u} = 0$ and the constraints given by the boundary conditions (33), where A is a certain set of paths in the configuration space of Navier–Stokes equations. For example, A can be a set of transition paths from the base flow to the stable traveling wave solution. Solving problem (36) is much more difficult than the deterministic Navier–Stokes equations. First of all, the Euler–Lagrange equation of problem (36) is a PDE of differentiation order 4 in space and of differentiation order 2 in time. Second, the scale of integration time T is of $O(Re)$. Simply speaking, solving problem

(36) is equivalent to solve a boundary-value problem, represented by a fourth-order nonlinear PDE, on a three-dimensional domain $[0, 2\pi] \times [-h, h] \times [0, T = O(Re)]$. Even for two-dimensional Navier–Stokes equations, we cannot afford the cost without parallel computing. Since T is large, the parallelization in time direction is particularly important.

The minimum action method for small random perturbations of Navier–Stokes equations was developed in [18]. The discretization of the action functional is based on spectral discretization in the physical space and *hp* finite element discretization in the time direction. The constraints from the divergence-free condition and the boundary conditions are removed by carefully choosing the approximation space such that the optimization problem becomes an unconstrained one. A simple diagonal preconditioner can also be found in [18].

For a certain range of Reynolds numbers and wave numbers, the plane Poiseuille flow has two stable solutions: one is the base flow $(\mathbf{u}_b, p_b)$ and the other one is a non-attenuated traveling wave $(\hat{\mathbf{u}}, \hat{p})$. Since we are mainly interested in the scalability of the hybrid code in this paper, we consider the following simple problem without loss of generality and more results about the transitions between the base flow and the stable traveling wave solution can be found in [18]. We take one snapshot of the traveling wave $(\hat{\mathbf{u}}(t_0), \hat{p}(t_0))$ at t_0, and use it as the initial condition for a dynamic solver of the Navier–Stokes equations to evolve time T. We then use $(\hat{\mathbf{u}}(t_0), \hat{p}(t_0))$ and $(\hat{\mathbf{u}}(t_0 + T), \hat{p}(t_0 + T))$ as the two ends of the transition path, in other words, we consider the following optimization problem

$$S_T(\mathbf{u}^*, p^*) = \min_{\substack{\mathbf{u}(0)=\hat{\mathbf{u}}(t_0),\ p(0)=\hat{p}(t_0) \\ \mathbf{u}(T)=\hat{\mathbf{u}}(t_0+T),\ p(T)=\hat{p}(t_0+T)}} S_T(\mathbf{u}, p). \tag{37}$$

In other words, the set A is defined as

$$A = \{\text{All transition paths on} [0, T] \text{ from } (\hat{\mathbf{u}}(t_0), \hat{p}(t_0)) \text{ to } (\hat{\mathbf{u}}(t_0 + T), \hat{p}(t_0 + T)).\}$$

Since there exists dynamics between $(\hat{\mathbf{u}}(t_0), \hat{p}(t_0))$ and $(\hat{\mathbf{u}}(t_0 + T), \hat{p}(t_0 + T))$, the minimal action path should be consistent with the trajectory of the traveling wave, along which the action functional is equal to zero. Then we can use the trajectory given by a dynamic solver of Navier–Stokes equations to verify the results given by the minimum action method.

In Fig. 7, we plot the strong scaling of the hybrid code for the discretization with 2048 linear finite elements in the time direction, 10 Fourier modes in the x direction and 32 Legendre modes in the y direction, where eight threads are used on each computation node and the strong scaling is with respect to the number of computation nodes. It is seen that a good scalability is obtained. Compared to the SODE case, the better strong scaling for the SPDE case is mainly due to the facts that (1) for each component of the gradient, the linear operator $\hat{\mathcal{G}}$ needs to be evaluated, which makes the evaluation of the gradient the most time-consuming part for each nonlinear CG iteration; (2) the finite element discretization in the time direction makes the computation of the gradient an element-wise operation, where OpenMP is effective since all local operations in each element share the information about the operator $\mathcal{G}$. For this test, we switched off the adaptivity in the time direction since it is problem dependent as discussed before.

Remark 2. For this problem, the hybrid strategy will be more effective in the sense that we can only use up to 2048 cores for a pure MPI strategy since we have 2048 time elements while the hybrid code can be implemented theoretically on $2048 \times 8 = 16384$ cores.

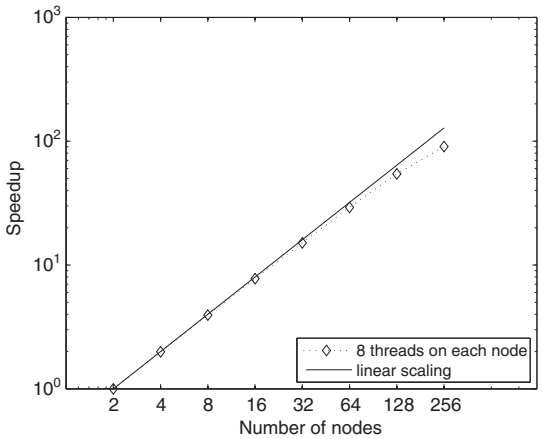

Fig. 7. Strong scaling of the hybrid code for two-dimensional channel flow. 2048 linear elements are used for the discretization in time direction, 10 Fourier modes are used for the x direction and 32 Legendre modes are used for the y direction. The total number of unknowns is 1162696. The scaling is with respect to the number of nodes, where on each node the number of threads are fixed. The speedup is computed with respect to the solution time given by the two-node simulations.

Remark 3. Since the discretized action functional is a nonlinear non-negative function, there must exist at least one minimum. However, this function is, in general, not convex. It is possible that the optimization solver will converge to a local minimum depending on the initial guess. In this sense, we need to couple the MAP with the study of the deterministic dynamical system for verification or for further improvement by choosing a more appropriate initial guess. We refer to [16] for more discussions about this issue.

5. Summary

In this work we present a hybrid (MPI/OpenMP) parallel strategy for the minimum action method to deal with small random perturbations of dynamical systems. Numerical experiments show that the developed hybrid algorithm has a good strong scalability. The main bottleneck for the scalability is from the time mesh adjustment. To alleviate such a difficulty, we need to enhance the parallel efficiency for the projection from the old time mesh to the new time mesh. A possible strategy is that we can solve the optimization problem using a sequence of time meshes where the time mesh will be refined gradually. Then the added time elements will be mainly local to each MPI process and will not introduce too many inter-node communications. In this work, we do not consider the parallelization of the nonlinear differential operator in space, which can be another bottleneck for spatially extended dynamical systems. The study of these issues is in progress.

Acknowledgments

This work was supported by Applied Mathematics program of the US DOE Office of Advanced Scientific Computing Research. X. Wan also acknowledges support from NSF grant DMS-1115632. The parallel computation was implemented on supercomputers supported by the Louisiana Optical Network Institute (LONI).

References

[1] C. Cerjan, W. Miller, On finding transition states, J. Chem. Phys. 75 (6) (1981) 2800–2806.
[2] W. E, W. Ren, E. Vanden-Eijnden, String method for the study of rare events, Phys. Rev. B 66 (2002) 052301.
[3] W. E, W. Ren, E. Vanden-Eijnden, Minimum action method for the study of rare events, Commun. Pure Appl. Math. 57 (2004) 637–656.
[4] W. E, W. Ren, E. Vanden-Eijnden, Simplified and improved string method for computing the minimum energy paths in barrier-crossing events, J. Chem. Phys. 126 (2007) 164103.
[5] W. Faris, G. Jona-Lasinio, Large fluctuations for a nonlinear hear equation with noise, J. Phys. A: Math. Gen. 15 (1982) 3025–3055.
[6] M.I. Freidlin, A.D. Wentzell, Random Perturbations of Dynamical Systems, 2nd Edition., Springer-Verlag, New York, 1998.
[7] W. Hager, H. Zhang, A new conjugate gradient method with guaranteed descent and an efficient line search, SIAM J. Optim. 16 (1) (2005) 170–192.
[8] M. Heymann, E. Vanden-Eijnden, The geometric minimum action method: a least action principle on the space of curves, Commun. Pure Appl. Math. 61 (2008) 1052–1117.
[9] G. Henkelman, H. Jónsson, A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives, J. Chem. Phys. 111 (15) (1999) 7010–7022.
[10] H. Jònsson, G. Mills, K. Jacobsen, Nudged elastic band method for finding minimum energy paths of transitions, in: B. Berne, G. Ciccotti, D. Coker (Eds.), Classical and Quantum Dynamics in Condensed Phase Simulations (1998).
[11] N. van Kampen, Stochastic Processes in Physics and Chemistry, North-Holland, 1981.
[12] G. Karniadakis, S. Sherwin, Spectral/hp Element Methods for Computational Fluid Dynamics, second ed., Oxford University Press, 2005.
[13] L. Onsager, S. Machlup, Fluctuations and irreversible processes, Phys. Rev. 91 (1953) 1505–1512.
[14] X. Zhou, W. Ren, W. E, Adaptive minimum action method for the study of rare events, J. Chem. Phys. 128 (2008) 104111.
[15] X. Zhou, W. E, Study of noise-induced transitions in the Lorenz system using the minimum action method, Commun. Math. Sci. 8 (2) (2010) 341–355.
[16] X. Wan, X. Zhou, W. E, Study of the noise-induced transition and the exploration of the configuration space for the Kuramoto–Sivashinsky equation using the minimum action method, Nonlinearity 23 (2010) 475–493.
[17] X. Wan, An adaptive high-order minimum action method, J. Comput. Phys. 230 (2011) 8669–8682.
[18] X. Wan, A minimum action method for random perturbations of two-dimensional parallel flows, J. Comput. Phys. 235 (2013) 497–514.