# Exploiting Task-Based Parallelism in Bayesian Uncertainty Quantification

Panagiotis E. Hadjidoukas[1(✉)], Panagiotis Angelikopoulos[1], Lina Kulakova[1], Costas Papadimitriou[2], and Petros Koumoutsakos[1]

[1] Computational Science and Engineering Laboratory, ETH Zürich, Zurich, Switzerland
{phadjido,pangelik,kulina}@mavt.ethz.ch, petros@ethz.ch
[2] Department of Mechanical Engineering, University of Thessaly, Volos, Greece
costas@uth.gr

**Abstract.** We introduce a task-parallel framework for non-intrusive Bayesian Uncertainty Quantification and Propagation of complex and computationally demanding physical models on massively parallel computing architectures. The framework incorporates Laplace asymptotic approximations and stochastic algorithms along with distributed numerical differentiation. Sampling is based on the Transitional Markov Chain Monte Carlo algorithm and its variants while the optimization tasks associated with the asymptotic approximations are treated via the Covariance Matrix Adaptation Evolution Strategy. Exploitation of task-based parallelism is based on a platform-agnostic adaptive load balancing library that orchestrates scheduling of multiple physical model evaluations on computing platforms that range from multicore systems to hybrid GPU clusters. Experimental results using representative applications demonstrate the flexibility and excellent scalability of the proposed framework.

**Keywords:** Task-based parallelism · Bayesian uncertainty quantification

## 1 Introduction

Computational models for scientific and engineering problems are developed based on the application of first principles, conservation laws and expert knowledge. Recent technological advances in sensing, measurement and imaging technologies provide an unprecedented opportunity to assist model development with an abundance of data. Data driven model discovery and evaluation of their predictive capabilities as in the context of Uncertainty Quantification and Propagation (UQ+P) is currently a topic of renewed interest [1]. Fusing both expert knowledge and experimental observations, Bayesian inference stands amongst the prevalent UQ+P techniques. It is used for quantifying and calibrating uncertainty models, as well as propagating these uncertainties in engineering simulations to achieve updated robust predictions of system performance, reliability and safety [2]. Common computational tools for performing Bayesian UQ+P


J.L. Träff et al. (Eds.): Euro-Par 2015, LNCS 9233, pp. 532–544, 2015.
DOI: 10.1007/978-3-662-48096-0_41

include Laplace methods of asymptotic approximation [3] and stochastic algorithms such as Markov Chain Monte Carlo (MCMC) and its variants [4].

Bayesian UQ+P tools involve global optimization problems, sampling from probability distributions, as well as evaluating high dimensional integrals. The computational challenge of Bayesian tools is the large number of model evaluations required, specifically in cases of complex engineering models with high resources requirements and time to solution. The need for multiple model evaluations leads, on average, to long turn-around time for Bayesian analysis, limiting its applicability when swift decisions are needed as in e.g. the case of earthquake early warnings system [5]. The ability to efficiently harness available computational resources is paramount for the Bayesian UQ+P framework and defines its applicability in engineering problems. The situation can be improved by advancing the computational efficiency of the models and by developing efficient UQ+P algorithms and computational frameworks that exploit massively parallel computing architectures. The focus of this paper is the latter.

A small number of parallel software frameworks for uncertainty quantification studies are currently available to the scientific community, with a non-exhaustive list containing: DAKOTA [6], PSUADE [7] and QUESO [8]. The parallelization of these systems has been mostly based on MPI and either follows a master-worker approach or applies domain decomposition to construct processor groups where simulations are assigned for execution. Most systems exploit only one level of parallelism, otherwise they rely on cumbersome implementations that apply hard partitioning of processing units. In addition, they lack runtime support for asynchronous nested task-based parallelism and adaptive load balancing and they do not take into account heterogeneous computing architectures. Consequently, they cannot counteract the increasing number of sources of load imbalance, such as variable processing power and simulation time, hardware and software faults and the irregularity of UQ algorithms.

We present a Bayesian computational framework for UQ that aims to address the above mentioned issues. The framework is based on the TORC task-parallel library for clusters [9], which is designed to provide unified programming and runtime support for computing platforms that range from single-core systems to hybrid multicore-GPU clusters and heterogenous Grid based supercomputers. Within this framework, we implement population based MCMC methods, the Transitional Markov Chain Monte Carlo (TMCMC) [10], Approximate Bayesian Computational Subset-Simulation (ABC-SubSim) [11], while the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [12] is used as an optimization tool. Note that all the algorithms implemented have highly parallel task graphs and thus are ideally suited for distributed and parallel computing.

## 2 Bayesian Formulation

In the Bayesian framework [13], the uncertainty in a parameter set $\underline{\theta} \in R^n$ of a model class $M$ simulating an engineering system is first quantified using a prior probability distribution function (PDF) $\pi(\underline{\theta}|M)$ and then updated using the

Bayes theorem to compute the posterior PDF $p(\underline{\theta}|D, M)$ based on available measurement data $D$ as: $p(\underline{\theta}|D, M) = p(D|\underline{\theta}, M)\pi(\underline{\theta}|M)/p(D|M)$ where $p(D|\underline{\theta}, M)$ is the likelihood of observing the data from the model class and $p(D|M)$ is the evidence of the model class. Assuming that the model predictions $\underline{g}(\underline{\theta}|M)$ and the measurement data $D = \{\underline{\hat{y}}\}$ satisfy $\underline{\hat{y}} = \underline{g}(\underline{\theta}|M) + \underline{e}$, where the prediction error term $\underline{e}$ , accounting for measurement, computational and modeling errors, is normally distributed with zero mean and covariance matrix $\Sigma$, the likelihood $p(D|\underline{\theta}, M)$ is given by [2] $p(D|\underline{\theta}, M) = |\Sigma(\underline{\theta})|^{-1/2}(2\pi)^{-n/2} \exp\left[-\frac{1}{2}J(\underline{\theta}; M)\right]$ where $J(\underline{\theta}; M) = [\underline{\hat{y}} - \underline{g}(\underline{\theta}|M)]^T \Sigma^{-1}(\underline{\theta})[\underline{\hat{y}} - \underline{g}(\underline{\theta}|M)]$, $|\cdot|$ denotes determinant, and the parameter set $\underline{\theta}$ is augmented to include parameters that are involved in the model structure of the correlation matrix $\Sigma$.

Bayesian computational tools include of stochastic algorithms and asymptotic approximations. Stochastic algorithms include variants of the MCMC technique [14] that are used to draw samples from the posterior PDF. TMCMC allows for the efficient execution of a large number of full system simulations on heterogeneous clusters/computers as described in Sect. 3, and can capture complex posterior PDFs. Using the Bayesian central limit theorem for large amounts of data, the posterior distribution of the model parameters can be asymptotically approximated by a Gaussian distribution centered at the most probable value $\hat{\underline{\theta}} = \text{argmin}_{\underline{\theta}}\, L(\underline{\theta}, M)$ of the model parameters, obtained by maximizing the posterior PDF $p(\underline{\theta}|D, M)$ or equivalently minimizing the function $L(\underline{\theta}; M) = -\ln p(\underline{\theta}|D, M) = \frac{1}{2}J(\underline{\theta}; M) - \ln \pi(\underline{\theta}|M)$ with covariance matrix equal to the inverse of the Hessian of the function $L(\underline{\theta}, M)$ evaluated at the most probable value $\hat{\underline{\theta}}$.

The asymptotic approximations for Bayesian model parameter and evidence estimation involve the solution of an optimization problem and the calculation of a single Hessian matrix [3]. Regarding the Hessian calculations, finite difference approximations of the gradient of the objective function scale up the computational effort by a factor proportional to the number of uncertain parameters. Computations can be performed in parallel since the derivatives of the objective function can be executed simultaneously, leaving the Time-to-Solution (TTS) independent of the number of uncertain parameters. Herein, numerical derivatives are calculated when needed fully in parallel using the non-intrusive adaptive parallel numerical differentiation library [15]. Evolution strategies are highly parallel and among several classes of evolution algorithms, CMA-ES [12] has been shown not only to to converge fast in particular when searching for a single global optimum, but to have an easily parallelizable task graph due to its generation based updating. Herein a task-parallel version of the CMA-ES is used to solve the single-objective optimization problems arising in Laplace asymptotic approximations.

In some cases the likelihood is hard to formulate (e.g. in case of stochastic model $M$) or hard to evaluate. ABC algorithms then are used to approximate the likelihood function $p(D|\underline{\theta}, M)$. A major difference of ABC algorithm as compared with standard Bayesian techniques is that it considers model parameters $\underline{\theta}$ and model outputs $\underline{x}$ as a pair of random variables and aims at evaluating the joint posterior distribution $p((\theta, \underline{x})|D, M)$. This can be done by applying

Bayes theorem and the chain rule: $p((\underline{\theta},\underline{x})|D,M) \propto p((\underline{\theta},\underline{x})|M)p(D|(\underline{\theta},\underline{x}),M) = p(\underline{\theta}|M)p(\underline{x}|\underline{\theta},M)p(D|(\underline{\theta},\underline{x}),M)$. The function $p(D|(\underline{\theta},\underline{x}),M)$ has a smaller discrepancy when the outcomes $\underline{x}$ are closer to the data $D$. ABC algorithms replace the equality with an approximation: $\underline{x} \approx D$. If $\underline{x}$ and $D$ are from a high-dimensional space, we introduces a vector of summary statistics $\eta(\cdot)$ to facilitate an easier comparison. The discrepancy between data and model outcome is then given by $\rho(\eta(\underline{x}),\eta(D))$ where $\rho(\cdot,\cdot)$ is some metric. An approximate joint posterior is defined for a tolerance level $\delta$ as $p_\delta((\underline{\theta},\underline{x})|D,M) \propto p(\underline{\theta}|M)p(\underline{x}|\underline{\theta},M)p(\rho(\eta(\underline{x}),\eta(D)) \leq \delta|(\underline{\theta},\underline{x}),M)$ where $\delta$ controls the quality of the posterior. Using the approximate posterior defined above, an ABC algorithm can evaluate the approximate joint posterior by simulating $\underline{\theta} \sim p(\underline{\theta}|M)$ and $\underline{x} \sim p(\underline{x}|\underline{\theta},M)$ and accepting the generated pair $(\underline{\theta},\underline{x})$ if $\rho(\eta(\underline{x}),\eta(D)) \leq \delta$.

## 3 Software and Runtime Environment

Aiming at support of both low-cost desktop machines and HPC environments from our Uncertainty Quantification and Optimization framework, we opted for a parallelization approach that:

- offers efficient exploitation of multilevel task-based parallelism
- provides ease of programming, hiding low-level parallelization details and thus facilitating algorithm development
- supports load balancing transparent to the user
- is highly portable and platform-agnostic, adapting automatically to the underlying hardware resources.

**The Task-Parallel Library.** In order to meet the above requirements, we based the parallel implementation of our tools on the TORC task-parallel library [9]. TORC provides a programming and runtime environment where parallel programs can be executed unaltered on both shared and distributed memory platforms. A TORC parallel application actually consists of multiple MPI processes that run on the cluster nodes and have one or multiple workers. Similarly to OpenMP, tasks are decoupled from the workers and thus the library allows for arbitrary nesting of tasks. Each worker continuously dispatches and executes tasks, submitted for execution to a set of priority queues. There is a single set of such queues in each MPI process and tasks are submitted to the queue that corresponds to the nesting level of parallelism they belong to. Task and data management are performed asynchronously and transparently to the user, by utilizing a server thread in each MPI process. The user can query the execution environment, e.g. number of workers, and specify the local or remote queue where each task will be submitted for execution. Due to the task stealing mechanism, idle workers can steal and execute tasks that have been submitted to a remote queue. An idle worker always try first to extract work from the lowest-level non-empty local queue. If there is no work available, it tries to steal tasks from the remote processes but starting from the highest-level queues. Therefore, the programmer is responsible for the task distribution policy: typically,

this involves cyclic distribution of first-level tasks among the workers and local submission of inner-level tasks. Combined with task stealing, this policy favors stealing of coarse-grain tasks and local execution of deeper levels of parallelism. In the context of this work, task functions receive as input an evaluation point, i.e. a set of parameters, and return a value computed at that point. The function can either include source code supplied by the user or invoke an external simulation program. The injected user code can embrace intra-node parallelism expressed with OpenMP directives or TORC tasks. Launching of external software is based on the fork-exec system calls while input data and results are communicated through the local filesystem. We do not pose any restrictions on the external software, which can be sequential or parallel. If the execution time of simulations is not high enough to hide the overhead of the launching procedure, a proxy process is created at program initialization for each worker. This process remains active throughout program execution, minimizing the spawning overhead by running directly the simulations. In addition, a persistent communication path based on Unix-domain sockets is established between each worker-proxy pair, minimizing the communication overheads.

When the application is executed with one process and multiple workers, the library operates exclusively through hardware shared memory avoiding message passing. TORC has been successfully used to provide runtime support to OpenMP and extensions of it on clusters.

**TMCMC.** A brief sketch of the TMCMC algorithm [10] is depicted in Algorithm 1. At the initialization stage, the algorithm selects randomly $C_1$ points which will serve as starting points for the MCMC chains for the first generation of the algorithm. The posterior evaluation for each point can be performed in parallel, while each evaluation can require a fixed number ($N_r$) of simulations. If $N_r > 1$ then the initialization exhibits two levels of parallelism that can be fully exploited. Each generation (TMCMC stage) $G$ involves the processing of $C_G$ MCMC chains of variable length, according to the statistics for the set of accepted points produced by the previous generation. As chains do not perform the same number of steps, load imbalance is introduced in the algorithm. They are instantiated with tasks and distributed appropriately to the workers, trying to balance the total workload among them without relying exclusively on the task stealing mechanism of TORC. At each step of a chain, the algorithm requires a posterior evaluation, which in turn may involve multiple independent simulation runs that are submitted for asynchronous execution as tasks. This exploitation of second-level parallelism provides more effective utilization of hardware resources as the higher number of tasks increases concurrency, resulting in better load balancing and reduced idle time for the workers.

The task stealing mechanism is essential for the efficient management of the irregular task parallelism exhibited by TMCMC. This irregularity is attributed to the variable numbers of chains per generation and steps per chain. The complexity of dealing with this irregularity becomes significantly higher if the execution time of model evaluations varies. In many cases the execution time cannot be estimated

**Algorithm 1.** TMCMC

```
1  Algorithm TMCMC()
       // Initialization
2      θ = {}
3      for each randomly selected starting point c = 1, ..., C_1 do
4          Compute function value F(c) = Posterior (c);
5          add c, F(c) to the set θ
6      end
7      compute statistics for the function values of the set θ
       // Main loop
8      for each generation g = 2, ..., G do
9          select C_g starting points from the set θ
10         θ = {}
11         for each chain c = 1, ..., C_g do
12             for each step s = 1, ..., S_c do
13                 propose next point p
14                 Compute function value F(c, s) = Posterior (p);
15                 accept/reject p, if accepted add it to the set θ
16             end
17         end
18         compute statistics for the function values of the set θ
19     end
20     return ;
21 Function Posterior(point p)
22     for t = 1, ..., N_r do
23         perform model evaluation M(p, t)
24     end
25     combine the results and compute F(p)
26     return F(p);
```

beforehand because it strongly depends on the input parameters of the search space where TMCMC is applied. Moreover, the execution time depends on the processing power of the underlying hardware, which can exhibit significant variability on computing environments that utilize heterogeneous nodes and hybrid computing architectures. TORC offers a programming and runtime environment where the irregular nested parallelism of TMCMC can be easily expressed and exploited at all possible levels, without making any assumption about the target hardware platform.

**Subset Simulation for Approximate Bayesian Computational.** Approximate Bayesian Computation Subset Simulation, *ABC-SubSim*, outlined in Algorithm 2, uses MCMC to efficiently generate conditional samples to gradually trace a rare event region. ABC-SubSim applies the idea of Subset Simulation to a special case of Approximate Bayesian Computation. The structure of ABC-SubSim is identical to that of TMCMC and differs in the following point: all MCMC chains in SubSim always perform the same predefined number of steps, in contrast to TMCMC where chain lengths are determined at runtime.

**Asymptotic Approximation.** The CMA-ES algorithm [12] includes, at each generation, a set of function evaluations that can be performed concurrently. The parallelization of CMA-ES using TORC is straightforward and involves the cyclic distribution of the tasks to the available workers. A second level

**Algorithm 2.** Subset Simulation

```
   // Initialization with Random Sampling from Prior
 1 z = {}
 2 for each randomly selected starting point c = 1, ..., C_1 do
 3 |   Compute function value F(c) = Prior (c);
 4 |   add c, F(c) to the set z
 5 end
 6 sort and keep the first a% of the set z, set discrepancy for next generation
   // Main loop
 7 for each generation g = 2, ..., G do
 8 |   select C_g starting points from the set θ
 9 |   z = {}
10 |   for each chain c = 1, ..., C_g do
11 |   |   for each step s = 1, ..., S do
12 |   |   |   accept/reject directions and propose next point p
13 |   |   |   Compute function value F(c, s) = Prior (p);
14 |   |   |   accept/reject p, if accepted add it to the set z, calculate acceptance rate ρ
15 |   |   end
16 |   end
17 |   sort and keep the first a% of the set z, set discrepancy for next generation, if ρ < 5 %
   |   then exit algorithm
18 end
```

of parallelism can be activated only if the objective function invokes multiple simulation runs, while load balancing issues arise on heterogeneous computing platforms or for variable execution time of simulation runs. The evaluation of the Hessian matrix is central to the Bayesian asymptotic approximation. This is normally provided as output of an optimization methodology (CMA-ES in our case). To select the appropriate differentiation step for each problem parameter, we first spawn tasks that compute partial derivatives for several differentiation steps. Then, we apply a Romberg extrapolation methodology to find the step with the most accurate result for each parameter and finally we compute the Hessian matrix. The multiple function evaluations introduce an additional level of task parallelism in the gradient and Hessian calculations, exploited by a parallel numerical differentiation module that has been also built on top of TORC.

## 4 Applications

In this section, we exemplify the key features and assess the parallel performance of our framework by performing UQ studies of representative applications. In particular, we compare the time to solution as well as the computational cost and PDF estimation efficiency for two engineering applications requiring significant computational resources. These applications exhibit significant TTS for a single posterior evaluation and target multi-core and hybrid CPU/GPU clusters. Furthermore, they demonstrate the coupling of third-party parallel scientific software into our framework.

### 4.1 TMCMC and CMA-ES on a GPU Cluster

We perform UQ+P in the most widely used MD model, that of water. We use a 5-site water model, TIP5P-E. The calibration data consist of the radial distribution

function of oxygen-oxygen in bulk water and its experimental uncertainty. Each evaluation of a posterior sample requires two full MD-simulation run, with the MD-code GROMACS 5.0 compiled with hybrid CPU-GPU acceleration. The final posterior value is computed by applying a post-processing stage which invokes a Matlab script that processes the output of the simulation run. The prediction error matrix $\Sigma$ can be decomposed into three contributions with elements $\Sigma_{ii} = \sigma_{exp}^2 + \sigma_{ens}^2 + \sigma_m^2$. We estimate the $\sigma_{ens}^2 \approx 0.005$. The experimental uncertainty contributions $e^{exp}$ are known and finally, the additional model prediction error term $\sigma_m^2$ is left to be determined from the inference process [16]. The parameters $\left(\epsilon_{\mathrm{O-O}}^{\mathrm{LJ}}, \sigma_{\mathrm{O-O}}^{\mathrm{LJ}}\right)$ and $q_{\mathrm{O}}$ are the Lennard-Jones interaction parameters and charge interaction respectively. We use truncated Gaussian priors for the three parameters with mean values based on the literature values for TIP5P [17], with a standard deviation of 30 % of $\bar{\theta}_\pi$, whereas the hyperparameter follows a Gamma prior, that is $\sigma_m^2 \sim \Gamma(1.5, 0.25)$.

**Results.** We present the timings and the results of the calibration of the TIP5-P water model. We performed our simulations on 32 compute nodes of the Piz Daint Cray XC30 cluster at the Swiss National SuperComputing Center CSCS. Each node is equipped with an 8-core Intel Xeon E5-2670 processor and one NVIDIA Tesla K20X GPU. TORC is initialized with a single worker per node because each single posterior evaluation task fully utilizes a compute node by means of the hybrid CPU/GPU configuration of GROMACS. Posterior evaluations are invoked by a separate proxy server process that receives a set of parameters, invokes the GROMACS model executions, the Matlab-based post-processing phase and finally sends back the posterior value. This approach, depicted in Fig. 1, minimizes runtime overheads because the Matlab environment is initialized only once and, furthermore, it offers high flexibility and portability.

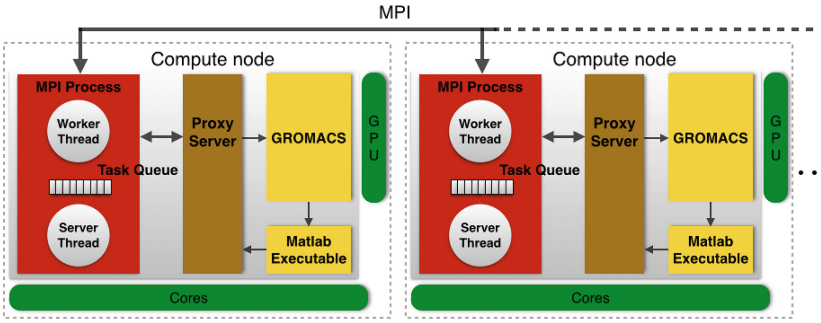


**Fig. 1.** Mapping of the parallel application on the compute nodes of the hybrid CPU/GPU cluster. The MPI application, the proxy server and the Matlab code run only on the cores while GROMACS is compiled with hybrid CPU/GPU configuration. To avoid initialization overheads, the Matlab code was compiled to a dynamic library and linked to the proxy process, replacing the Matlab executable depicted, for simplicity reasons, in the figure.

Each posterior evaluation requires between 17 and 21 min of wall clock-time in the above mentioned computing architecture. The variation of the mean time for completing each posterior evaluation is due to the different runtime for different initial parameters. The variance in the evaluation time and the maximum chain length are the main sources of load imbalance in this application. We address the first issue by using 256 samples per generation, i.e. 8x the number of workers, while we alleviate the second problem by sorting the chains according to their length and then evenly distributing the total workload to the available workers. The maximum chain length determines the lowest possible processing time for each generation and the maximum number of workers above which execution time does not improve and parallel efficiency is negatively affected.

Figure 2 (top, left) depicts the efficiency of TMCMC, while Fig. 2 (top, right) depicts how the time of a single posterior evaluation varies over a total of 15 generations. The above solutions, along with the stealing mechanism of TORC, minimize the idle time of workers and result in parallel efficiency higher than 97 % when every worker executes the same number of posterior evaluations. The lower efficiency (≈88.4 %) for the 12th and 14th generation of TMCMC is attributed to the fact that the maximum chain length was equal to 9 for both cases, which imposes an upper limit of 88 % to the expected efficiency. Similar behavior is observed in Fig. 2 (bottom) for the parallel CMA-ES, where parallel efficiency and statistics for the evaluation time are reported every 10 generations. We notice that the measured parallel efficiency is equal to 90.1 % at the end of the 10th generation, which is due to the lower number of samples (64) per generation and the high variance of the evaluation time. This variance decreases as the algorithm evolves and the efficiency increases accordingly up to 97.4 %.

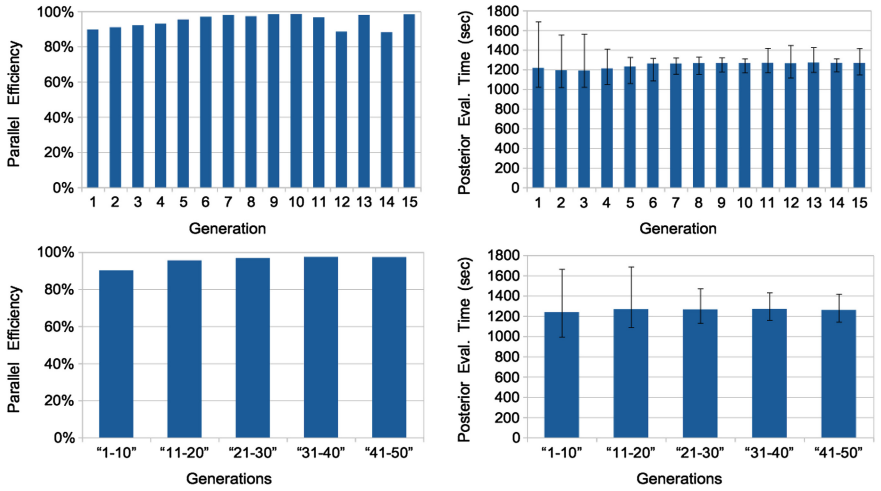


**Fig. 2.** Top: Parallel efficiency of TMCMC (left) and posterior evaluation time for the MD simulation (right). Bottom: Corresponding figures for CMA-ES.

**Table 1.** Computational effort of the MD calibration.

| Method | Generations | Samples | Evaluations | TTS (hours) | Overall efficiency |
|---|---|---|---|---|---|
| TMCMC | 14 | 256 | 3584 | ≈44.0 | 94.5 % |
| CMA-ES | 50 | 64 | 3200 | ≈36.7 | 95.4 % |

**Table 2.** Mean values and Coefficient of Variation of the posterior distribution of the model parameter, along with the LogEvidence values of each model class.

| Class | $\bar{\epsilon}^{LJ}_{O-O}$ | $u_{\epsilon^{\mathrm{LJ}}_{\mathrm{O-O}}}$ | $\bar{\sigma}^{LJ}_{O-O}$ | $u_{\sigma^{\mathrm{LJ}}_{\mathrm{O-O}}}$ | $\bar{q}_O$ | $u_{q_O}$ | $\bar{\sigma}^2_m$ | $u_{\sigma^2_m}$ | LogEvidence |
|---|---|---|---|---|---|---|---|---|---|
| TMCMC | 0.688 | 4.3 % | 0.312 | 0.16 % | 0.2417 | 0.76 % | 0.00923 | 15.1 % | 1401.34 |
| CMA-ES | 0.651 | 4.3 % | 0.313 | 0.20 % | 0.2392 | 0.81 % | 0.01034 | 16.0 % | 1414.21 |

The computational cost of the MD calibration with the two methods is presented in Table 1. The mean parameter estimates as well as their associated uncertainty are summarized in Table 2. The coefficient of variation $u_\theta$ of a parameter $\theta$ is defined as the sample standard deviation of that parameter over its estimated mean $\bar{\theta}$.

### 4.2 ABC-Subsim on a Multicore Cluster

As a stochastic model we took the calibration of the Lennard-Jones potential parameters for helium. To perform the calibration we used the data on the Boltzmann factor $f_B = \left\langle \exp\left(-\frac{H}{k_B T}\right)\right\rangle$ where $H$ is the enthalpy of the system of helium atoms, $T$ is the temperature of the system, $k_B$ is the Boltzmann constant and $\langle\cdot\rangle$ denotes the ensemble average. The data was generated using the software LAMMPS for a system of 1000 atoms for 20 ns in the NPT ensemble with a timestep of 2fs. The system used for calibration consists of 1000 atoms and is equilibrated for 2ns, following a production run in the $NPT$ ensemble for another 2ns with a 2fs timestep. We performed calibration with 2 different settings. 1) Assuming the resulting Boltzmann factor distribution was Gaussian, and a discrepancy function of: $\rho(x,y) = \sqrt{((\mu_x - \mu_y)/\mu_x)^2 + ((\sigma_x - \sigma_y)/\sigma_x)^2}$. In the second setting the discrepancy is the given: $\rho(x,y) = D_{KL}(P||Q)$ where $D_{KL}$ is the Kullback-Leibler divergence, $P$ is the data distribution, $Q$ is the simulation outcome distribution of the Boltzmann factor.

**Results.** The algorithm runs a full molecular dynamic simulation for every parameter set and hence requires a significant amount of computational work. It also exhibits two levels of parallelism, as the Markov chains with different seeds can be processed in parallel while each single simulation can also run in parallel using the MPI version of LAMMPS.

The time to solution for each function evaluation varies with the given parameters, introducing load imbalance in the algorithm. We deal with this issue by

submitting tasks with higher execution time first: we sort the samples according to the value of the $\sigma$ parameter before distributing the corresponding function evaluation or Markov chain tasks to the workers. Moreover, we enhance this scheme with the task stealing of TORC.

We performed our simulations on 512 compute nodes of the Piz Daint cluster (4096 cores in total). TORC is initialized with two MPI workers per node and each LAMMPS simulation utilizes 4 cores in turn. The population size was set to be 15360 and the Markov chain length was equal to 5. The algorithm stops when the acceptance rate drops below 5 %.

Table 3 summarizes the parallel performance of ABC-SubSim. Despite the high variance of the time for a single simulation run, we observed that the efficiency of the initialization phase (level 0) reaches 82 % as 15360 function evaluations are distributed among the 1024 workers. The lower efficiency (70.5 %) of Level 1 is attributed to the existence of chains with high accumulated running times and the small number of available chains that correspond to each worker (3072 chains in total, 3 chains per worker). As the algorithm evolves, the efficiency increases and reaches 92 % for the last level, which exhibits a load imbalance of approximately 8 % as computed by $(T_{max} - T_{avg})/T_{avg}$, where $T_{max}$ and $T_{avg}$ are the maximum and average time that the workers were busy during the processing of the specific level. The information about the prior and the posterior values of the parameters is given in Table 4.

**Table 3.** Detailed per-level performance results of ABC-SubSim on 512 nodes of Piz Daint. $T_f$ shows the mean and standard deviation of the simulation times and $T_w$ is the wall-clock time per generation, respectively. All the times are reported in seconds.

| Level | $T_f$ | $T_w$ | Efficiency |
|---|---|---|---|
| 0 | 82 ± 83 | 1497 | 81.8 % |
| 1 | 87 ± 57 | 1843 | 70.5 % |
| 2 | 68 ± 10 | 1237 | 81.9 % |
| 3 | 65 ± 6 | 1110 | 88.4 % |
| 4 | 66 ± 5 | 1078 | 92.2 % |

**Table 4.** Prior and posterior information of parameters of the Helium system in molecular LAMMPS units. The number of generations $N_{gen}$ computed before the acceptance rate reached a threshold value of 5 % and achieved tolerance levels $\delta$ for two models: $M_G$ [Gaussian setting], $M_{KL}$ [Kullback-Leibler setting]. Prior bounds $[\theta_l, \theta_r]$, mean values $\bar{\theta}$ and coefficients of variation $u_\theta$ of the Lennard-Jones parameters of Helium.

| Model | $[\sigma_l, \sigma_r]$ | $\bar{\sigma}$ | $u_\sigma$ | $[\epsilon_l, \epsilon_r]$ | $\bar{\epsilon}$ | $u_\epsilon$ | $N_{gen}$ | $\delta$ |
|---|---|---|---|---|---|---|---|---|
| $M_G$ | [0.1,0.8] | 0.2452 | 11.5 % | [0.01,1.0] | 0.423 | 64.5 % | 4 | $3.40\times10^{-3}$ |
| $M_{KL}$ | [0.1,0.8] | 0.2792 | 5.0 % | [0.01,1.0] | 0.117 | 15.4 % | 6 | $6.70\times10^{-2}$ |

## 5 Conclusions

We presented a computational framework for large scale Bayesian uncertainty quantification and stochastic optimization that can exploit massively parallel and hybrid (CPU/GPU) computing architectures. The framework incorporates several state-of-the-art stochastic algorithms for the computation of the likelihood that are capable of sampling from complex, multimodal posterior distribution functions. Built on top of the TORC task-parallel library, it offers straightforward extraction and exploitation of multilevel task-based parallelism in stochastic optimization and sampling algorithms. It targets both distributed and shared memory systems in a platform-agnostic way and provides transparent load balancing for efficient scheduling of multiple function evaluations. The Bayesian tools are written as clients upon the layer of the library and can be integrated with legacy codes as well as black-box system models, resulting in an easily extensible non-intrusive framework. The present framework aims to facilitate the implementation of UQ+P on engineering applications and harness the capabilities of contemporary and emerging HPC architectures. Current work includes the development of surrogate models and performance studies on the Intel Xeon Phi architecture.

Our open-source software can be downloaded from http://www.cse-lab.ethz.ch/software/Pi4U. We acknowledge computational time at the Swiss National Supercomputing Center CSCS under project number s448.

## References

1. Owhadi, H., Scovel, C., Sullivan, T., McKerns, M., Ortiz, M.: Optimal uncertainty quantification. SIAM Rev. **55**(2), 271–345 (2013)
2. Beck, J.L., Yuen, K.V.: Model selection using response measurements: Bayesian probabilistic approach. J. Eng. Mech. **130**(2), 192–203 (2004)
3. Papadimitriou, C., Beck, J.L., Katafygiotis, L.S.: Asymptotic expansions for reliability and moments of uncertain systems. J. Eng. Mech. **123**(12), 1219–1229 (1997)
4. Chen, M.H., Shao, Q.M., Ibrahim, J.G.: Monte Carlo Methods in Bayesian Computation. Springer, New York (2000)
5. Wu, S., Beck, J.L., Heaton, T.H.: Earthquake probability-based automated decision-making framework for earthquake early warning applications. Comp. Aid. Civ. Infr. Eng. **28**, 737–752 (2013)
6. Adams, B., Bohnhoff, W., Dalbey, K., Eddy, J., Eldred, M., Gay, D., Haskell, K., Hough, P., Swiler, L.: DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Sandia Technical report (2013)
7. Lawrence Livermore National Laboratory. The PSUADE UQ project. http://computation.llnl.gov/casc/uncertainty_quantification/
8. Prudencio, E., Cheung, S.H.: Parallel adaptive multilevel sampling algorithms for the Bayesian analysis of mathematical models. Int. J. Unc. Quan. **2**(3), 215–237 (2012)
9. Hadjidoukas, P.E., Lappas, E., Dimakopoulos, V.V.: A runtime library for platform-independent task parallelism. In: 20th International Conference on Parallel, Distributed and Network-Based Processing, pp. 229–236 (2012)

10. Ching, J.Y., Chen, Y.C.: Transitional markov chain Monte Carlo method for Bayesian model updating, model class selection, and model averaging. J. Eng. Mech. **133**(7), 816–832 (2007)
11. Chiachio, M., Beck, J., Chiachio, J., Rus, G.: Approximate Bayesian computation by subset simulation. SIAM J. Sci. Comput. **36**, A1339–A1358 (2014)
12. Hansen, N., Muller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comp. **11**(1), 1–18 (2003)
13. Beck, J.L., Katafygiotis, L.S.: Updating models and their uncertainties. I: Bayesian statistical framework. J. Eng. Mech. **124**(4), 455–461 (1998)
14. Galbally, D., Fidkowski, K., Willcox, K., Ghattas, O.: Non-linear model reduction for uncertainty quantification in large-scale inverse problems. Int. J. Num. Meth. Eng. **81**(12), 1581–1608 (2010)
15. Hadjidoukas, P.E., Angelikopoulos, P., Voglis, C., Papageorgiou, D.G., Lagaris, I.E.: NDL-v2.0: A new version of the numerical differentiation library for parallel architectures. Comput. Phys. Comm. **185**(7), 2217–2219 (2014)
16. Angelikopoulos, P., Papadimitriou, C., Koumoutsakos, P.: Data driven, predictive molecular dynamics for nanoscale flow simulations under uncertainty. J. Phys. Chem. B **117**(47), 14808–14816 (2013)
17. Rick, S.: A reoptimization of the five-site water potential (TIP5P) for use with Ewald sums. J. Chem. Phys. **120**, 6085–6093 (2004)