Proceedings of Parallel Scientific Computing
ID5130
Jan-May, 2024, Chennai, India

IMPLEMENTATION OF MONTE CARLO METHODS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Jay Jitendra Prajapati
ME21B143
Department of Mechanical Engineering
Indian Institute of Technology Madras

Shrey Rakeshkumar Patel
ME21B138
Department of Mechanical Engineering
Indian Institute of Technology Madras

ABSTRACT

Partial Differential Equations (PDEs) represent fundamental mathematical models in numerous scientific and engineering disciplines, yet their solution remains a formidable challenge due to inherent complexities. Traditional numerical methods encounter limitations in handling high-dimensional systems, intricate boundary conditions, and nonlinearities inherent in many practical scenarios. Monte Carlo methods offer a potent alternative, leveraging stochastic simulation to approximate solutions to PDEs. This paper provides a thorough investigation into the utilization of Monte Carlo methods for PDE solving, with a specific focus on their parallel implementation utilizing OpenMP, MPI and OpenACC frameworks. It begins by elucidating the intrinsic difficulties associated with PDEs, emphasizing the inadequacies of conventional numerical techniques. Subsequently, Monte Carlo methods are introduced as versatile tools capable of addressing the aforementioned challenges by employing random trajectories or paths to approximate solutions through extensive sampling.

Various Monte Carlo techniques tailored for solving different types of PDEs are examined in detail. Special attention is given to methodologies such as the Monte Carlo Finite Difference method and the Random Walk method, highlighting their efficacy in handling complex PDEs with irregular geometries and boundary conditions. Furthermore, the paper explores the potential of parallel computation using OpenMP, MPI and OpenACC to enhance the scalability and efficiency of Monte Carlo simulations. By distributing computational tasks across multiple processors or computing nodes, parallelization enables significant improvements in computational speed and the ability to tackle larger and more intricate PDEs within practical time constraints.

Finally, the performance of these algorithms in the serial and parallel implementations is compared. Future advancements in parallel algorithms and hardware architectures are anticipated to further augment the performance and applicability of Monte Carlo methods, fostering innovation and breakthroughs in computational science and engineering.

INTRODUCTION

I. MONTE CARLO METHODS

Monte Carlo Methods are a class of computational solutions that can be applied to vast ranges of problems which rely on repeated random sampling to provide generally approximate solutions [1], [2]. Monte Carlo is a stochastic approach, in which series of simulations (trials), representing the analysed problem, with randomly selected input values, are performed. Among these trials, a specified number of properly defined successes is achieved. The ratio between the number of success trials to the number of all trials, scaled by dimensional quantity (e.g., area, function value) allows for the estimation of the unknown solution, providing the number of trials is large enough.

They can be understood as procedures for solving non probabilistic problems by probabilistic type methods [3].

They are used in cases where analytical or deterministic numerical. In deterministic models, a particular state (either continuous or discrete) is uniquely assigned to input data and no element of randomness occurs. On the other hand, the probabilistic class of mathematical models specifies governing differential equations that combine one or more random variables (with assigned probability distribution function) with other non-random variables.

II. POISSON AND LAPLACE EQUATION

Poisson equation is one of the most important equations in applied mathematics and has applications in such fields as astronomy, heat flow, fluid dynamics, and electromagnetism and describes stationary (time-independent) problems such as torsional deflection of a prismatic bar, stationary heat flow, distribution of electrical potential or filtration through porous media [4]. It is an elliptic partial differential equation which describes equilibrium and is a part of Boundary Value Problems (BVP). It takes the general form of:

$$\bigtriangledown^2 \varphi = f \; ; \; \varphi(x) = g(x)$$

Here,φ is a scalar field, f is any given function of space and g is the boundary conditions. In steady state conduction with heat generation problems, temperature distribution satisfies Poisson's equation, which is an excellent example of an elliptic partial differential equation. In two dimensions, Poisson's equation can be written as

$$\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y} = f(x,y)$$

The Laplace Equation is a special case of the Poisson Equation where f is known to be zero [5]. This is the case of calculating the electric potential when there are no charges in the region. The Laplace Equation has the advantage of having a computationally easier solution.

A. *Procedure for solving the Poisson Equation with Fixed step Monte Carlo Method*

1) To apply MCM to the poisson equation, we first have to understand the concept of a random walk (RW). Consider a rectangular lattice that covers the domain of Ω with φ each lattice point at a distance h from its neighbours. Starting from a point A in this lattice, we move randomly to one of the closest neighbours of the point in each step, with equal probability.

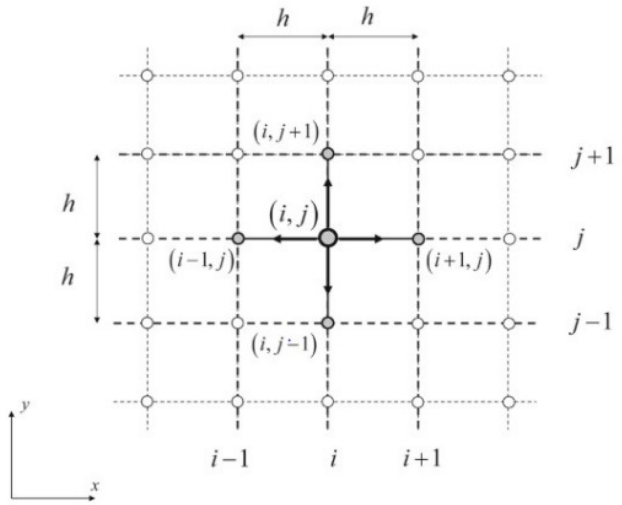

FIGURE 1. Lattice in a 2D space. Beginning from point (i, j) we randomly move to any of its closest four neighbours. Source: [6]

2) We repeat this type of movement until we reach the boundary point a of the domain at which point the random walk w concludes. Compute a term F as

$$F(w) = \sum_{i \in interior\ points\ in\ random\ walk} \frac{f(i)h^2}{4}$$

3) Repeat this RW for a finite number of times (N) and calculate the probability of the walk starting at A and concluding at each boundary point a as $P_A(a)$.

$$P_A(a) = \frac{Number\ of\ times\ random\ walk\ ends\ at\ a}{N}$$

4) Using the known values of boundary condition g at each boundary point a compute the quantity $u(A)$ as

$$u(A) = \sum_{a \in boundary\ points} g(a)P_A(a) - \sum_{for\ all\ w} F(w)$$

5) Calculate u for all points in the domain. This is the approximate solution to the differential equation.

$$\varphi(A) = u(A) \quad A \in Lattice\ points$$

B. ***Procedure for solving the Poisson Equation with Variable step Monte Carlo Method***

1) Consider a rectangular lattice that covers the domain of Ω with φ each lattice point at a distance h from its neighbours. Starting from a point A, designated (x_i, y_i), in this lattice, find the shortest distance r_i to the nearest boundary point.

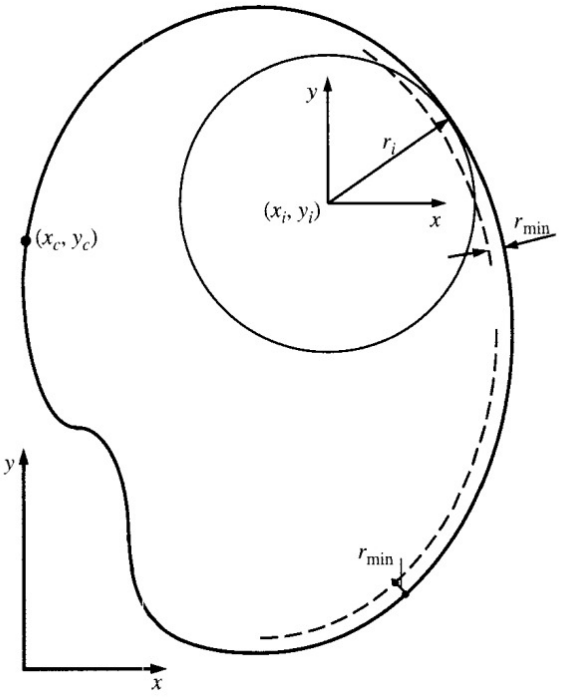

FIGURE 2. Schematic representation for a two-dimensional floating random walk. Source: [7]

2) Then, a random number ϕ is selected form a set of uniformly distributed random numbers between 0 and 2π. The random number gives the angular direction in which a random walker will move.The coordinate of the new location of the random walk, designated are (x_{i+1}, y_{i+1}), are

$$x_{i+1} = x_i + r_i \cos(\phi_i) \qquad y_{i+1} = y_i + r_i \cos(\phi_i)$$

3) The next step of random walk is executed by using (x_{i+1}, y_{i+1}) as center and computing the shortest distance between this new center and the nearest section of the boundary.
4) This process continues until the random walk is terminated; that is, when radius r_i of a circle is smaller than a preassigned value r_{min} and the floating random walk can be regarded as being located at the nearest point on the boundary. The value of r_{min} should be sufficiently small on the order of $\frac{h}{4}$.
5) We repeat this type of movement until the random walk w terminates and compute the ternm F as

$$F(w) = \sum_{i \in interior\ points\ in\ random\ walk} \frac{f(i) r_i^2}{4}$$

6) Repeat the same procedure as fixed step for calculating values of lattice points.

III. 2D TRANSIENT EQUATION

Parabolic PDEs arise in a multitude of natural and engineered systems characterized by diffusion, heat conduction, chemical reactions, and other transient phenomena. Understanding the dynamics governed by these equations is crucial for predicting and controlling the behavior of such systems. From heat transfer in materials to population dynamics in ecology, from financial modeling to image processing, parabolic PDEs find widespread application across diverse fields of science and engineering.

The following represents the transient equation in 2D:

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y}\right)$$

A. ***Procedure for solving the Transient Equation with Fixed step Monte Carlo Method***

1) Consider a rectangular lattice that covers the domain of Ω with φ each lattice point at a distance h from its neighbours.
2) Starting from point A at time t, designated as (x_i, y_i, t) in this lattice, we move randomly to one of the points from time $t - \triangle t$ with probability given below:

$$P_{x_{i+1}, y_i} = P_{x_{i-1}, y_i} = P_{x_i, y_{i+1}} = P_{x_i, y_{i-1}} = \frac{\alpha \triangle t}{h^2} \quad (1)$$

$$P_0 = 1 - \frac{\alpha \triangle t}{h^2} \quad (2)$$

where 1 represents the probability of choosing nearest neighbours at time $t - \triangle t$ and 2 represents the probabilistic of choosing same point at time $t - \triangle t$. [7]

3) Continue this process until random walk is terminated; that is, we reach the known boundary condition.
4) Repeat this RW for a finite number of times (N) and calculate the probability of the walk starting at A and concluding at each boundary point a as $P_A(a)$.

$$P_A(a) = \frac{Number\ of\ times\ random\ walk\ ends\ at\ a}{N}$$

5) Using the known values of boundary condition g at each boundary point a compute the quantity $u(A)$ as

$$u(A) = \sum_{a \in boundary\ points} g(a) P_A(a)$$

6) Calculate u for all points in the domain. This is the approximate solution to the differential equation.

$$\varphi(A) = u(A) \quad A \in Lattice\ points$$

B. ***Procedure for solving the Transient Equation with Variable step Monte Carlo Method***

1) Consider the same lattice and starting from point A, designated (x_i, y_i, t),in this lattice, find the shortest distance r_i to the nearest boundary point.
2) Then a random number ϕ is selected from a set of uniformly distributed random numbers between 0 and 2π.
3) The random number gives the angular direction in which a random walker will move.The coordinate of the new location of the random walk, designated are $(x_{i+1}, y_{i+1}, t - \tau)$, are

$$x_{i+1} = x_i + r_i \cos(\phi_i) \qquad y_{i+1} = y_i + r_i \cos(\phi_i)$$

4) The probability function $\mathscr{Z}$ provides the probable time duration associated with each step given as below:

$$\mathscr{Z}(\frac{\alpha\tau}{r_i^2}) = 1 - 2\sum_{k=0}^{\infty} \frac{exp(-\lambda_k^2 \alpha\tau / r_i^2)}{\lambda_k J_1(\lambda_k)}$$

in which the λ_k values are the roots of Bessel function $J_0(\lambda)$. Source: [8]

5) Generate a random number, between 0 and 1, assigned to $\mathscr{Z}$ and for given values of α and r_i, evaulate the value of τ.
6) This process continues until the random walk is terminated; that is, when radius r_i of a circle is smaller than a preassigned value r_{min} and the floating random walk can be regarded as being located at the nearest point on the boundary. The value of r_{min} should be sufficiently small on the order of $\frac{h}{4}$.
7) Repeat the same procedure as fixed step for calculating values of lattice points.

PARALLELIZATION STRATEGY

Let us consider a lattice covering the domain Ω, where each lattice point, denoted by φ, is situated at a distance h from its neighboring points. This lattice serves as the discretization framework for the domain, facilitating the analysis and computation of various properties within Ω.

The Monte Carlo method for solving partial differential equations offers two potential avenues for parallelization. The first level of parallelism involves traversing grid points, enabling concurrent computation across the domain. Additionally, a second level of parallelism is attainable through the execution of multiple random walks at individual grid points, further enhancing computational efficiency and scalability.

During the execution of a random walk at a grid point, the potential for parallelization is constrained by data dependencies. Specifically, the evaluation of the next step is contingent upon the current step, thus precluding concurrent computation. As a result, parallelization efforts are limited in this context due to the inherent data dependency inherent in the random walk process. Close to the boundary points, the rate of convergence of the random walk is observed to be significantly higher compared to that of interior points.

Parallelization with OpenMP

In OpenMP, parallelization is achieved by distributing the grid points across multiple threads. This strategy enables concurrent computation of grid points, thereby leveraging the computational resources more effectively and potentially reducing execution time.

To address the challenges posed by boundary points, the OpenMP framework offers the schedule clause, allowing for the cyclic distribution of work among threads. This approach, often

referred to as block partitioning, ensures that each thread receives a balanced workload, mitigating the need for threads to wait for others to complete their tasks.

The depicted plots illustrate the speed-up achieved by the parallel solver for Poisson equation.

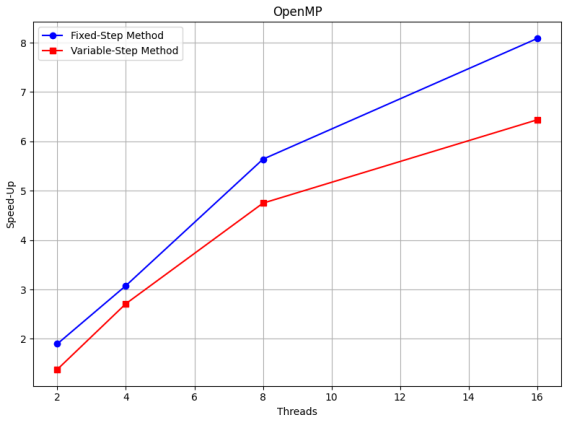

FIGURE 3. Speedup versus threads plot for $\Delta = 0.01$ and for 100 random walks

The depicted plots illustrate the speed-up achieved by the parallel solver for Transient equation.

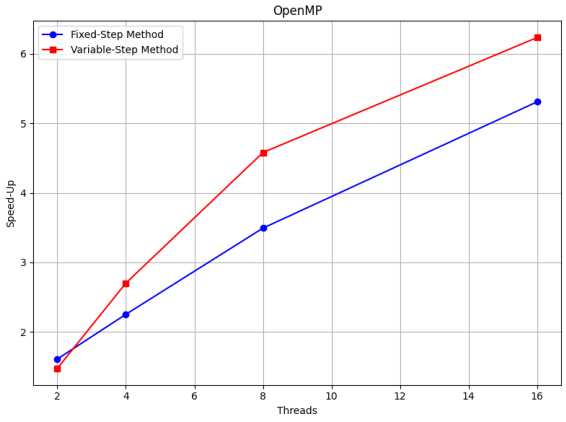

FIGURE 4. Speedup versus threads plot for $\Delta = 0.01$ and for 100 random walks

Parallelization with MPI

When considering the two available options for parallelization, it becomes evident that parallelizing across random walks would necessitate each process to allocate memory equivalent to the grid size. Moreover, the overhead associated with communication during the data gathering process at the host process is notably high. Consequently, it is preferable to parallelize across grid points rather than random walks, as this approach minimizes memory requirements and mitigates the impact of communication overhead, thereby enhancing overall computational efficiency.

In MPI, parallelization is accomplished by partitioning the grid points among different processes in a cyclic manner. This strategy enables each process to independently operate on a subset of the grid, facilitating parallel computation and leveraging distributed memory architectures for enhanced performance. Upon completion of the computation, all local variables across every process are gathered at the host process for subsequent post-processing tasks. This consolidation of data allows for comprehensive analysis and synthesis of results, enabling the extraction of meaningful insights from the distributed computations.

The depicted plots illustrate the speed-up achieved by the parallel solver for Poisson equation.

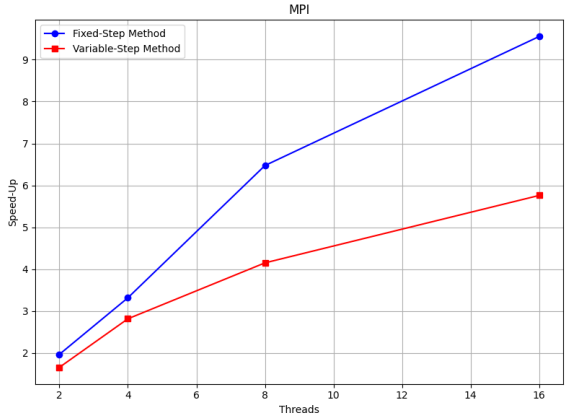

FIGURE 5. Speedup versus processes plot for $\Delta = 0.01$ and for 100 random walks

The depicted plots illustrate the speed-up achieved by the parallel solver for Transient equation.

Parallelization with OpenACC

In the context of OpenACC, the consideration of two potential levels of parallelization reveals a limitation in fully exploiting

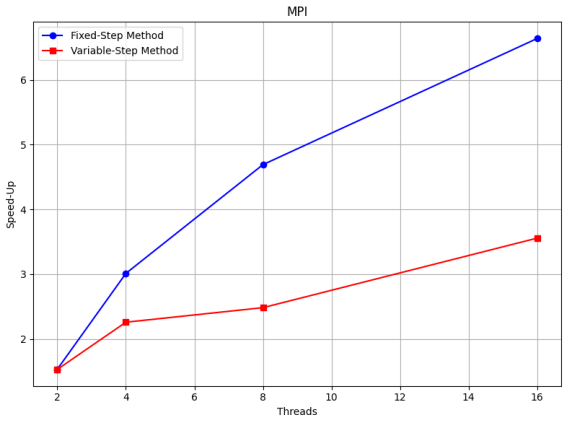

FIGURE 6. Speedup versus processes plot for $\Delta = 0.01$ and for 100 random walks

the hierarchical levels of parallelism inherent to the framework. This constraint arises due to the dependency of pseudo-random numbers on previous values, rendering it impractical to leverage nested parallel constructs within OpenACC.

Parallelization in the context of grid points is achievable through the distribution of workloads across gangs and workers within OpenACC. Additionally, to minimize overheads, an initial step involves copying all essential variables from the host to the device. This proactive measure facilitates efficient computation by ensuring that necessary data is readily accessible on the device, thereby mitigating communication overheads associated with data transfers during runtime.

The depicted plots illustrate the speed-up achieved by the parallel solver for Poisson equation.

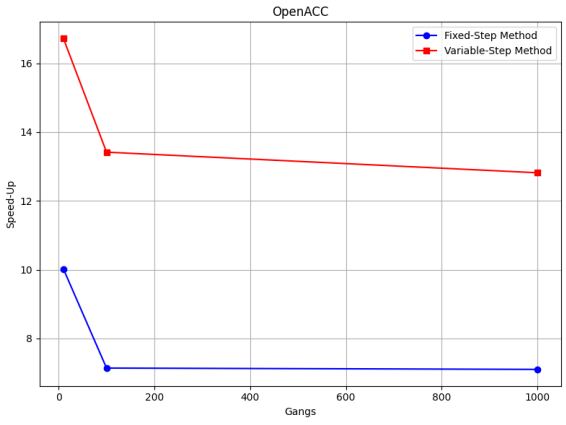

FIGURE 7. Speedup versus gangs plot for $\Delta = 0.01$ and for 100 random walks

The depicted plots illustrate the speed-up achieved by the parallel solver for Transient equation.

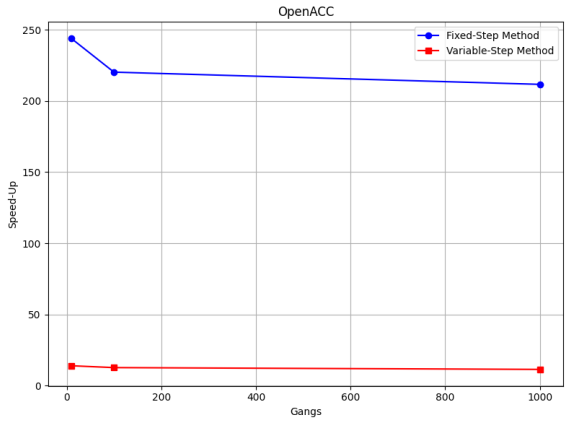

FIGURE 8. Speedup versus gangs plot for $\Delta = 0.01$ and for 100 random walks

RESULTS AND DISCUSSION

1. The investigation into the solution of the Poisson equation, as illustrated in Figures 3, 5, and 7, delineates a discernible contrast in computational performance between fixed step and variable step methods when implemented alongside parallel computing frameworks. Specifically, the analysis reveals that fixed step methods exhibit greater computational efficiency, as evidenced by their notable speedup with OpenMP and MPI. Conversely, variable step methods demonstrate enhanced computational acceleration, particularly evident through their superior speedup when integrated with OpenACC.
2. Looking at how we solve transient equations, shown in Figures 4, 6, and 8, we notice some interesting differences in performance between fixed and variable step methods when we use different parallel computing tools. It seems that when we use OpenACC and MPI, fixed step methods give us a better speed-up compared to variable step methods. On the other hand, when we use OpenMP, variable step methods seem to be provide better speed-up than fixed step methods.
3. In both the Transient and Poisson equations, we noticed that as we increased the number of threads or processes in OpenMP and MPI setups, the speedup also increased consistently. However, when it comes to OpenACC, we found that increasing the gang sizes didn't always lead to a proportionate increase in speedup. In fact, we observed a decrease in speedup as the gang sizes grew larger.

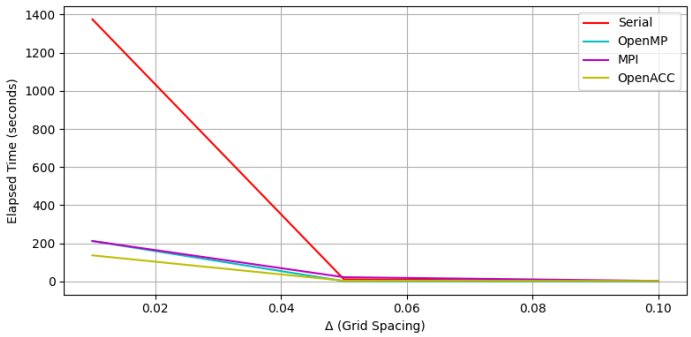

FIGURE 9. plot of elapsed time vs grid spacing for Poisson equation with fixed step for 100 random walks

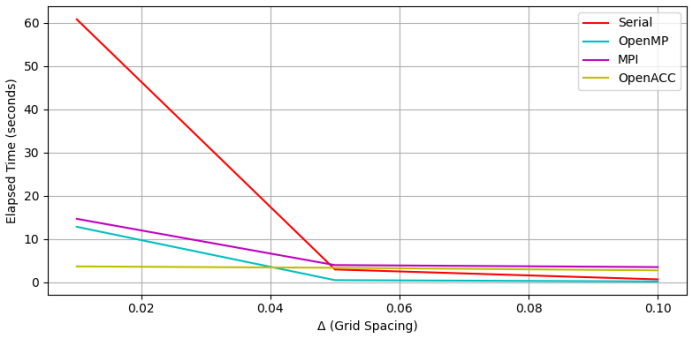

FIGURE 10. plot of elapsed time vs grid spacing for Poisson equation with variable step for 100 random walks

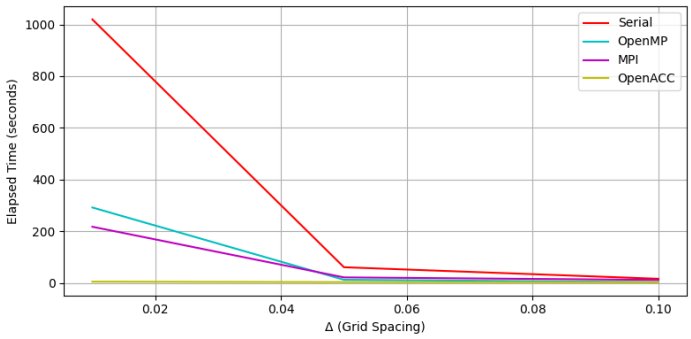

FIGURE 11. plot of elapsed time vs grid spacing for Transient equation with fixed step for 100 random walks

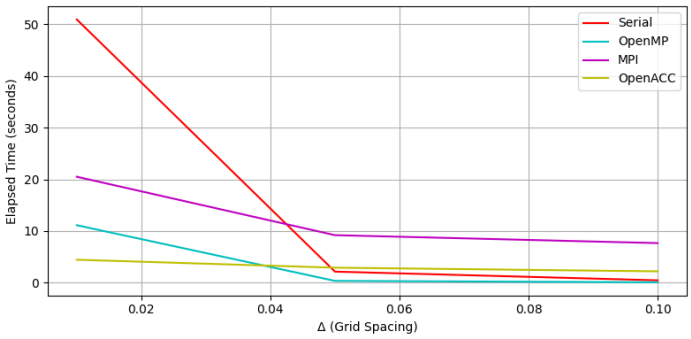

FIGURE 12. plot of elapsed time vs grid spacing for Transient equation with variable step for 100 random walks

4. Based on the observations derived from Figures 9, 10, 11, and 12, it is evident that the variable step method consistently outperforms the fixed step method in terms of computational efficiency for both equations under consideration.
5. In scenarios involving a smaller number of grid points, the superior performance of OpenMP compared to other parallel and serial computing frameworks stems from its utilization of shared memory architecture, which is particularly advantageous in such contexts.
6. In scenarios characterized by a large number of grid points, OpenACC exhibits superior performance over all other parallel and serial frameworks due to its adept utilization of many-core architectures.
7. In cases involving fewer grid points, distributed memory architectures perform inadequately in contrast to shared memory architectures, primarily due to the considerable communication overhead outweighing the actual computation cost. Conversely, when the computational workload surpasses the communication overhead, OpenACC and MPI demonstrate superior performance.

CONCLUSION

The resolution of partial differential equations stands as a fundamental endeavor across numerous domains. Monte Carlo methods offer a viable approach for tackling such equations, presenting ample opportunities for parallelization. We have effectively devised and implemented a parallel algorithm leveraging Monte Carlo methods for solving partial differential equations. Notably, our investigations reveal that OpenACC exhibits superior performance for larger grid sizes, while OpenMP proves more efficient for smaller grid sizes.

ACKNOWLEDGMENT

Thanks go to Dr. Kameswararao Anupindi for their support.

REFERENCES

[1] Wikipedia, Accessed 2024. Monte carlo method. Retrieved from Wikipedia.

[2] Mascagni, M. “Monte carlo methods for partial differential equations: A personal journey”. PhD thesis, Florida State University.

[3] Ali, A. A., Ullmann, E., and Hinze, M., 2017. "Multilevel monte carlo analysis for optimal control of elliptic pdes with random coefficients". *SIAM/ASA Journal on Uncertainty Quantification,* ***5***(1), pp. 466–492.
[4] Wikipedia, Accessed 2024. Poisson's equation. Retrieved from Wikipedia.
[5] Wikipedia, Accessed 2024. Laplace's equation. Retrieved from Wikipedia.
[6] Milewski, S., 2020. "A matlab software for approximate solution of 2d elliptic problems by means of the meshless monte carlo random walk method". *Numerical Algorithms,* ***83***(2), February, pp. 565–591.
[7] Rohsenow, W. M., Hartnett, J. P., Cho, Y. I., et al., 1998. *Handbook of heat transfer*, Vol. 3. Mcgraw-hill New York.
[8] Haji-Sheikh, A., and Sparrow, E. M., 1966. "The floating random walk and its application to monte carlo solutions of heat equations". *SIAM Journal on Applied Mathematics,* ***14***(2), pp. 370–389.