2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)

Event-Triggered Communication in Parallel Computing

Soumyadip Ghosh[1], Kamal K. Saha[2], Vijay Gupta[1], and Gretar Tryggvason[3]

[1]Department of Electrical Engineering, University of Notre Dame
{sghosh2,vgupta2}@nd.edu
[2]Center for Research Computing, University of Notre Dame
ksaha@nd.edu
[3]Department of Mechanical Engineering, Johns Hopkins University
gtryggv1@jhu.edu

***Abstract*— Communication overhead in parallel systems can be a significant bottleneck in scaling up parallel computation. In this paper, we propose event-triggered communication methods to reduce such communication overhead for numerical simulation of partial differential equations. As opposed to traditional communication in which processing elements exchange data at every iteration of the numerical algorithm, the main idea behind event-triggered communication is to exchange data only when necessary as dictated by a suitably defined event. We show through numerical experiments that these methods have potential to reduce simulation time. Strong scaling plots show that the approach may be effective on large machines as well.**

I. INTRODUCTION

Communication is typically costlier in terms of time and energy than computation and has been identified as a bottleneck for scaling up numerical simulations on parallel computing systems [1]. In this paper, we are interested in addressing this problem in the context of numerical simulation of partial differential equations (PDEs). A straight-forward and popular approach is to discretize the spatial domain and assign various parts of the domain to different processing elements (PEs). Different PEs then execute the algorithm being used to solve the PDE numerically on its own assigned sub-domain. However, in a typical implementation, the PEs carry out two types of communication at every iteration:

- Local communication of the values of the physical variables on the boundary of their sub-domain to the PEs that are assigned neighboring sub-domains to ensure that the spatial derivatives are correctly calculated. This is popularly known as *halo exchange*.
- Global communication of the convergence criterion with every other PE to check if the specified tolerance of the numerical algorithm has been met on the entire domain. Computation of dot products for basis vectors in Krylov subspace methods also requires global communication.

Each such communication event is typically accompanied by a synchronization event that ensures that all the values needed by a PE from other PEs have been successfully received before it carries out its next iteration. In other words, even if a PE carries out its computations quickly, it must wait for the other PEs to complete their computations and communicate all values needed by it before it can proceed to the next iteration.

Reducing communication overhead in parallel computing systems has received much attention in literature. Two main approaches seem to have been proposed. One approach is that of *asynchronous* algorithms proposed in the context of local communication. These algorithms still carry out the local communication at every iteration, but relax the synchronization due to local communication. This means that the PEs still transmit data at every iteration for local communication, however they do not wait for the most updated values to be successfully received from the other PEs before starting the next iteration. Instead they proceed with the last values that were successfully received. Chazan [2] pioneered this direction in the 1960s by introducing chaotic relaxation, with further generalizations introduced by Baudet [3]. Bru [4] proposed new asynchronous methods by combining relaxation and matrix splitting techniques. For surveys of asynchronous methods, the reader is referred to Amitai [5] and Frommer [6], with a more recent survey by Jimack [7]. For the problem of interest to this paper, asynchronous methods have recently been applied to numerical solution of PDEs by Donzis [8] and Aditya [9].

The second approach is that of *communication-avoiding* (CA) algorithms that relax the requirement of communication at every iteration. Because of reduced communication, synchronization requirements are also reduced. In comparison to asynchronous algorithms which only save time, CA algorithms also save energy expended for data movement. Besides reducing communication between PEs, these algorithms have also been proposed for reducing communication between levels of memory hierarchy. A lot of work has been performed to relax the global communication in Krylov subspace methods. We can point to Van Rosendale [10] and Chronopoulos [11], who pioneered the development of *s*-step methods in Krylov subspace algorithms in which the basis vectors needed for orthogonalization are computed once every *s* steps. Toledo [12] proposed a way to optimize the computation of the *s*-step basis in the *s*-step conjugate gradient (CG) method so as to reduce the number of words transferred between levels of memory hierarchy. Hoemmen [13] proposed a CA-CG method that reduced communication both between levels of memory hierarchy and between processors in a network. Carson [14] studied the performance

978-1-7281-0176-7/18/$31.00 ©2018 IEEE

of the CA-CG method on the Hopper supercomputer using a simple Poisson model problem. Yamazaki [15] improved the performance of CA-GMRES on multicores with multiple GPUs. Examples of CA techniques applied to numerical algorithms other than Krylov subspace methods include Basu [16], who applied a compiler-optimized CA scheme to geometric multigrid, and Baboulin [17], who used a CA-partial pivoting scheme to speed up Gaussian Elimination. Ballard [18] designed a communication-optimal parallel algorithm for Strassen's matrix multiplication that achieves some theoretical lower bounds on communication costs. As a follow-up, Lipshitz [19] showed that the communication-avoiding parallel Strassen method proposed in [18] achieves better performance than tuned implementations of other classical matrix multiplication algorithms. Recently, Murthy [20] designed a CA compiler that analyzes data dependence patterns in a program to reduce communication.

Communication avoiding algorithms have, thus, emerged as powerful tools to reduce the communication overhead in parallel simulations. Abstractly, these algorithms can be viewed as relaxing the requirement to communicate at every iteration (i.e., time-triggered) to communicating only when a certain event is satisfied (i.e. event triggered). Thus, for instance, in the CA-CG method of Hoemmen [13], the event is the completion of s iterations, so that communication occurs once every s iterations. In this paper, we design a new class of CA algorithms in which the events that trigger local communication are dependent on the values in the local domain of the sending PE. The specific event that we consider in this paper is that communication happens when the boundary value being communicated by the sending PE has changed from the last communicated value by more than a user specified threshold. This condition is representative of the *state* of a PE. Such state dependent events can modulate the communication frequency according to how quickly the values at any PE are changing; thus, intuitively achieving better reductions in communication overhead than those achieved using state independent events (such as periodic communication [13]). However, communication with such state dependent events is aperiodic and largely unpredictable, thus requiring implementations with MPI one-sided, which are significantly more complicated. For completeness, we also consider communication being triggered by events that are not dependent on the state by considering periodic communication. While we choose numerical solution of partial differential equations (PDE) as our example, the techniques can be extended to other problems like optimization which also involve solving algebraic system of equations like PDEs. Similarly, while we focus on the case where the Jacobi method is being used for the sake of simplicity, extensions to other algorithms like Krylov subspace methods, geometric multigrid, etc. are possible. We confine ourselves to communication between PEs in a network rather than communication involved in accessing memory and show that the proposed event-triggered methods can effectively reduce the simulation time.

The paper is organized as follows :- Section II formulates the model PDE that we use for experiments described in the paper. Section III introduces the main idea behind our approach to CA algorithms with implementation details in Sections IV and V. Section VI concludes the paper with directions for future work.

II. PROBLEM FORMULATION

For the experiments described in this paper, we consider a 2D Poisson PDE used to express the electric potential resulting from a distribution of electric charges. In our simulations, we consider the charge distribution to be an electric dipole - a point positive charge and a point negative charge at specified locations in the domain. The PDE is given by

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = -\frac{\rho}{\epsilon}, \tag{1}$$

where V is the electric potential, ρ is the electric charge and ϵ is the permittivity of free space equal to $8.854 \times 10^{-12} m^{-3} kg^{-1} s^4 A^2$. Using the finite-difference method, the PDE in (1) is discretized as follows:

$$\frac{V_{i-1,j} + V_{i+1,j} - 2V_{i,j}}{\Delta x^2} + \frac{V_{i,j-1} + V_{i,j+1} - 2V_{i,j}}{\Delta y^2} = -\frac{\rho_{i,j}}{\epsilon}, \tag{2}$$

where $\Delta x, \Delta y$ are the grid resolutions and i, j are the grid indices in the two dimensions. For convenience, we consider a square 2D domain and set $\Delta x = \Delta y$. We consider Dirichlet boundary conditions - meaning the values along the boundaries at the four edges are maintained at a constant value throughout the numerical solution.

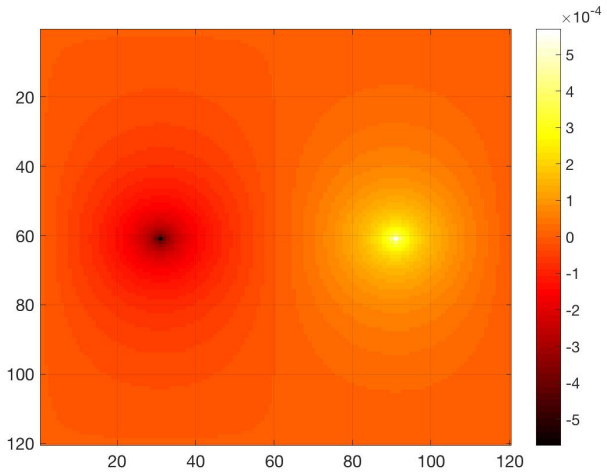

Fig. 1. Electric potential due to a dipole calculated after convergence when the numerical scheme discussed in Section II is implemented on a single PE. Positive charge of the dipole is located at (90,60) and the negative charge at (30,60)

To numerically solve (2), a variety of iterative techniques can be used. We choose the Jacobi method which is the simplest relaxation-type iterative solver for solving a system of equations. The numerical solver using Jacobi method is

given by a recurrence relation in which the values of the potential at every grid point at iteration $k+1$ are computed from the values at iteration k according to the relation

$$V_{i,j}(k+1) = \frac{1}{4}\Big(V_{i-1,j}(k) + V_{i+1,j}(k) + V_{i,j-1}(k) + V_{i,j+1}(k) + \frac{\Delta x^2 \rho_{i,j}}{\epsilon}\Big). \quad (3)$$

This solver thus sets up a fixed-point iterative problem. As a stopping condition for the numerical scheme, we use the residual at any iteration. The residual starts from a non-zero value and is monotonically non-increasing. For our simulations, we use the stopping criterion that the maximum value of the residual in the 2D domain is less than the tolerance of 10^{-5}. Fig 1 shows the resulting electric potential V after convergence when the scheme described above is implemented on a single PE.

For numerical solution of the PDE in a parallel setting, the domain needs to be decomposed first. We consider a 1D domain decomposition parallel to the horizontal axis. Decomposition of the domain leads to two forms of communication among the PEs. The first is the local communication involving the halo exchange where the PEs have to communicate the entire boundary row to the neighboring PE. The second is the global communication arising due to the requirement for monitoring the residual-based convergence criterion. In a traditional implementation, both these forms of communication would occur at every iteration k and may become bottlenecks in a large scale implementation. In Section III, we describe the main idea behind our proposed approach to relax these communication requirements. Detailed experimental results with the proposed approach are presented in Sections IV and V.

For all our simulations, we use an HPC cluster of nodes with each node having 2 CPU Sockets of AMD's EPYC 24-core 2.3 GHz processor and 128 GB RAM per node. The cluster uses Mellanox EDR interconnect. The MPI library chosen is mvapich2 built with Intel compiler. The length of the 2D domain is considered to be 1 unit and the grid resolution is chosen to be 5760 in both dimensions.

III. MAIN IDEA OF THE PROPOSED APPROACH

The main idea behind our approach is to relax the requirement to communicate at every iteration. Instead, we propose to trigger communication in events when a certain condition is satisfied. This condition can be quite general. In this paper, we study two types of conditions. In the first case, the event triggering the communication depends on the state of the PE, and in the second case, the event does not depend explicitly on the state of the PE. As a reminder, the state of the PE refers to the values of the grid points in the sub-domain allocated to the PE.

a) State dependent triggering events: The main intuition behind communication based on state-dependent events is to communicate 'only when necessary'. In other words, if the value of a variable has not varied much from the value last communicated, we can choose not to communicate, and instead, the intended recipient could use the last communicated value without incurring much error in its computation. While the event triggering the communication can be designed in a variety of ways, for specificity, in this paper, we design it based on the change in values (from the last communicated values) of the points on the boundary of the sub-domain allocated to the sending PE. When this change exceeds some user specified threshold, the values are communicated; otherwise, the intended receiver PEs use the values last communicated from the sender PE.

Remark 1: The boundary in the two-dimensional PDE that we consider in this paper is a row vector. The evolution of every grid point in this row may exceed the threshold criterion at different times. Rather than allowing every grid point along the boundary to start an MPI message of unit length independently, it is more efficient to monitor some norm of the values of the grid points of the entire boundary row and communicate the values for the entire row as a single large MPI message when the norm satisfies the event condition. We use the infinity norm of the vector, i.e., its maximum value, in this work.

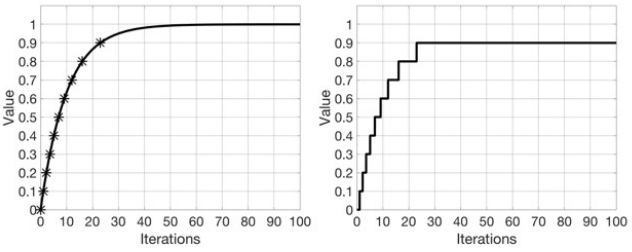

Fig. 2. The basic idea behind communicating only when the value of the boundary points changes by some threshold from the previously communicated value. The left plot shows the maximum value of the boundary vector at the sending PE. The asterisks show the points at which the threshold criterion triggers the communication. The right plot shows the values used by the receiving PE.

Fig 2 illustrates the basic idea discussed above considering a threshold of 0.1. The left plot shows the evolution of the maximum value of the boundary vector at the sending PE. When the value changes by more than 0.1 from the previously communicated values, a communication event is triggered as marked by an asterisk. The right plot shows the corresponding ghost row values in the receiving PE. When a communication event happens, the most current value is used by the receiving PE for computation. Otherwise, the last communicated value is used. Note that this leads to a staircase type evolution of the value of the ghost points at the receiving PE.

Remark 2: In the above discussion, we have focused on local communication among PEs. While the global communication to check if the convergence criterion has been met may also be made dependent on similar events, in this paper, we make the global communication independent of the state for simplicity. Instead, we consider the global communication to happen periodically, once every r iterations for various values of r, similar to that in the case of state-independent triggering events.

An algorithmic description of the proposed approach is given as Algorithm 1.

Algorithm 1 Communication Triggered by State Dependent Events

```
do
    Compute V_{i,j}(k+1) as in (3)
    if Change in boundary values of PE ≥ threshold then
        Communicate boundary values to neighbor(s)
    end if
    if k mod r == 0 then
        Compute Local residual
        Global residual ← max(All local residuals)
    end if
while Global residual > tolerance
```

Note that the event we have considered is quite basic. More sophisticated event choices may include adaptive thresholds based on the derivatives of boundary values. The ghost points in the recipient PE may also be extrapolated using history of last communicated values, instead of following a staircase evolution. We note that the concept of state-dependent communication described here has been proposed under the name of event-triggered communication in networked control systems [21] and wireless networks [22].

b) State-independent triggering events: As discussed more fully in Section IV, the implementation of state-dependent triggering events requires the use of MPI one-sided, which is significantly more complicated than the usual implementations. For comparison, we also consider a class of algorithms in which the triggering events do not depend on the values of a PE's sub-domain, i.e., the state of a PE. In this work, we focus on the algorithm in which local and global communication happens periodically with a user specified period r. Once again, the receiver PE uses the value that was last communicated to it in between iterations involving communication.

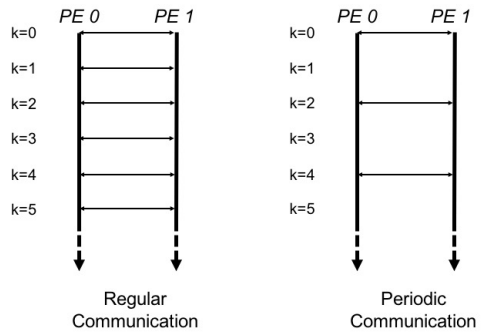

Fig. 3. Communication events when a periodic communication scheme is adopted for local communication. Period of two is shown here.

Fig 3 illustrates the timeline for local communication among PEs when the period is set to be two. The global communication to check the convergence criterion is also made periodic. This allows us to compare the performance of the algorithms when the local communication is triggered by events that are state-independent and events that are state-dependent.

Remark 3: Hoemmen [13] proposed a periodic CA algorithm similar to the one proposed above in the context of computation of basis vectors for the orthogonalization procedure in Krylov subspace methods which requires global communication. We propose that local communication be made periodic in addition to global communication and illustrate this on Jacobi iteration methods to solve PDEs numerically.

An algorithmic description of the proposed approach is given as Algorithm 2.

Algorithm 2 Communication Triggered by State Independent Events

```
do
    Compute V_{i,j}(k+1) as in (3)
    if k mod r == 0 then
        Communicate boundary values to neighbor(s)
        Compute Local residual
        Global residual ← max(All local residuals)
    end if
while Global residual > tolerance
```

IV. STATE-DEPENDENT TRIGGERING EVENTS

A. Implementation

Events that trigger communication depend on the state of the sending PE. Thus, when the sending PE wants to initiate a communication, it can issue an MPI_Send operation. However, since the receiving PE is not aware of the state of the sending PE, it does not know when to issue an MPI_Recv operation. Hence standard two-sided communication cannot be used to implement communication triggered by state-dependent events.

We select one-sided communication for this purpose [23]. In one-sided communication, only the sending PE has to know all the parameters of the message for both the sending and receiving side - the receiving PE plays no role (hence the alternate name of *Remote Memory Access (RMA)*). This property makes only one-sided communication suitable for our purpose since with state-dependent triggering events, communication is dependent only on the state of the sending PE without any involvement of the receiving PE.

Before the start of iterations in the numerical solver, every PE defines a region of memory called *window* using MPI_Win_allocate which is public, meaning that other PEs have permission to access this window. This window represents the ghost cells of a PE. So, when a communication event is triggered in the sending PE, it uses MPI_Put to write the message directly in the window of the receiving PE.

However, one-sided communication decouples data transfer from synchronization. Hence it is the programmer's responsibility to synchronize explicitly to ensure that RMA operations have completed. The MPI standard defines two

methods for this purpose. In *active target synchronization* based methods, all the PEs define a time window within which RMA access to any PE is permitted. This can be done through implementation of a collective concept of *fence* which signifies the beginning and end of an access epoch. All PEs collectively call MPI_Win_fence at the start and the end of RMA access. Between the two fences, all RMA operations in the entire communicator must complete. There is a more general form of active target synchronization where there is no collective concept of fence - instead only a subgroup of PEs synchronize by a post/start/complete/wait (PSCW) mechanism.

However, such active target synchronization requires MPI calls from the receiving PE. In our proposed approach, communication is dependent only on the state of the sending PE. Thus, the receiving PE is not aware of when a neighbor PE will send a message, and so it cannot invoke an MPI call. This makes active target synchronization unsuitable for our application.

Instead, we utilize *passive target synchronization*, that does not require any MPI calls from the receiving PE. The sending PE performs communication operations in access epochs demarcated by MPI_Win_lock and MPI_Win_unlock calls as illustrated in Fig 4. Note that the word "lock" is a misnomer here because these routines do not provide a traditional lock or mutex. MPI_Win_lock means that the origin PE can begin access on the target PE. Similarly, MPI_Win_unlock means that the origin PE can end access on the target PE. Since passive target synchronization does not require any MPI calls from the receiver, we utilize it for our implementation of state-dependent communication triggering events.

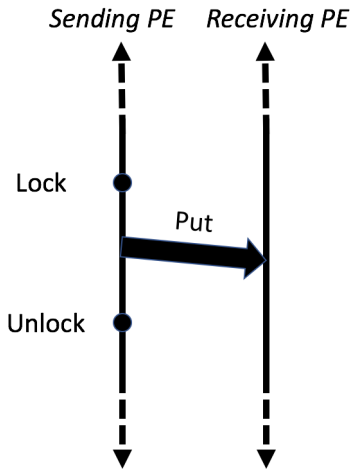

Fig. 4. Illustration of timeline of event-triggered local communication using passive target synchronization.

The above discussion focuses on local communication for halo exchange. For global communication of the convergence criterion, all the PEs need to exchange information among each other at the same time. Since every PE triggers events at different times based on its own state, there is no structured way to relax the MPI_Allreduce involving global communication here. While we have some initial ideas on how to make global communication triggering events state dependent as well, we do not implement them in this paper. Instead, global communication happens periodically with a user-specified period similar to that in the state-independent communication scheme described in Section V.

B. Numerical Stability

Stability of a numerical scheme implies approaching the fixed point solution of the discretized system of equations given in (3). We note that the Jacobi solver with communication in every iteration as in (3) is numerically stable. However, with state-dependent event-triggered communication, the final solution may be different depending on the value of threshold. To measure the error in the solution, we consider the solution with threshold 0 as the benchmark and compare the solution with any threshold against the benchmark solution. Note that the case of threshold 0 yields the original numerical scheme where communication happens at every iteration.

Fig 5 shows the log-log plot of the error from benchmark solution for various thresholds. Specifically, we quantify the error as the maximum of the absolute difference of the domain values for a particular threshold after convergence with the domain values of the benchmark solution after convergence. We see that for a certain range of low thresholds, the error stays the same, implying that the quality of solution is the same in this range. As threshold starts increasing, the error grows in order. This is explained by the fact that very few messages are exchanged in higher thresholds. In the region of very high thresholds, no communication happens between PEs, so they treat their sub-domain boundaries as local boundary conditions and converge in isolation. Hence the error from the benchmark solution starts approaching a limiting value.

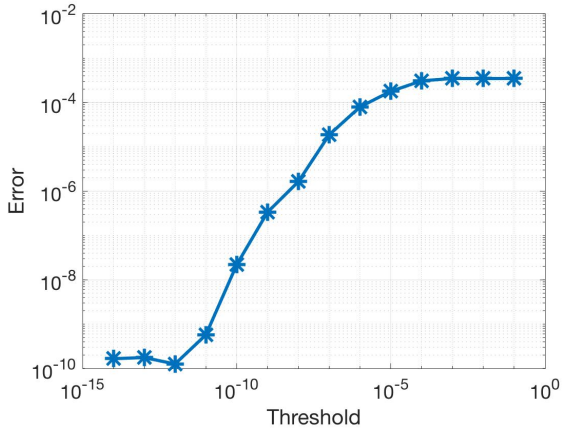

Fig. 5. Effect of threshold for event-triggered communication on error from benchmark solution.

C. Time Reduction

Fig 6 shows the reduction in time for simulations run with parameters already mentioned in Section II. Since convergence to the correct solution is dependent on the value of the threshold as stated above, we select the threshold to be 10^{-14}. Using this threshold, we observe the simulation time across various periods of global communication. Since the state-independent communication scheme in Section V also implements periodic global communication, we can compare the effect of just local communication in the state-dependent scheme in Fig 6 with that in the state-independent scheme in Fig 9 for the same period. We note that the time taken in Fig 6 is overall lesser than that in Fig 9.

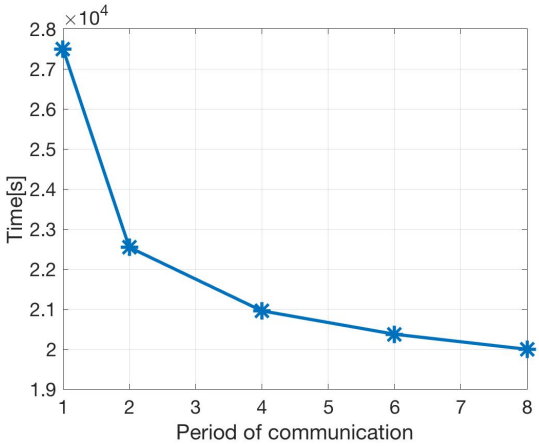

Fig. 6. Reduction in simulation time for state-dependent local communication across various periods of global communication.

D. Scaling

We show the strong scaling of this scheme for threshold 10^{-14} across various periods of global communication in Fig 7 and the corresponding speedup plot in Fig 8. These results are found to be better than the strong scaling of the state-independent communication scheme shown later in Fig 10 and Fig 11. Note that we run our simulations on multiples of entire nodes each of which has 48 cores.

V. STATE-INDEPENDENT TRIGGERING EVENTS

A. Implementation

Now we describe the implementation details of the state-independent periodic communication. The local communication due to halo exchange is done using standard MPI point-to-point communication where the transmitting PE posts an MPI_Send while the receiving PE posts an MPI_Recv. In order to do periodic communication, the MPI_Send and MPI_Recv are called periodically. Since the period is known to every PE before the start of the simulation, every PE knows when to call MPI_Send/MPI_Recv and when to skip it.

For global communication involving computation of the convergence criterion, MPI_Allreduce is also called with

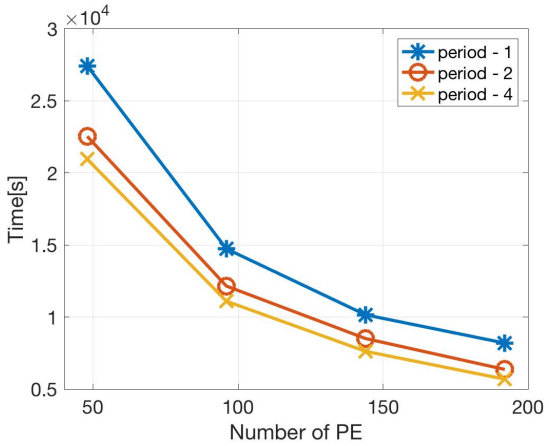

Fig. 7. Strong Scaling of state-dependent communication. The simulations are run on $\{48, 96, 144, 192\}$ PEs.

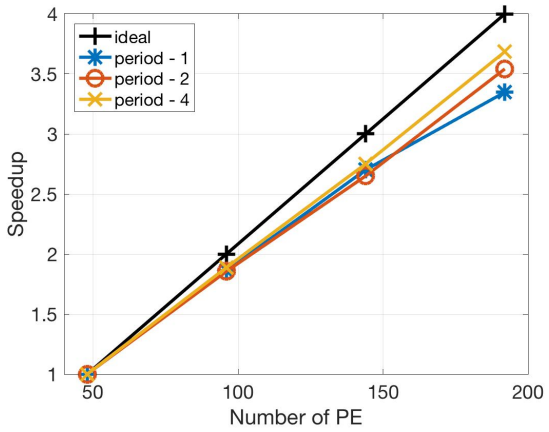

Fig. 8. Speedup of Strong Scaling results in Fig 7. Since we run our simulations in multiples of 48 PEs from 48 to 192, the ideal scaling ranges from 1 to 4.

the same period to perform the reduction operation on the residual across all the PEs and check for convergence.

B. Numerical Stability

In our previous work, we showed that the numerical scheme with periodic communication asymptotically converges to the same solution for any period under Dirichlet boundary conditions [24]. Note that a period of 1 is the traditionally implemented numerical scheme where both local and global communication happens at every iteration. So, for any value of period r, the numerical scheme asymptotically approaches the same solution.

C. Time Reduction

Fig 9 shows the reduction in time across different periods of local and global communication with the parameters already described in Section II. As the period of communication becomes higher, time is saved due to lesser communication. However, to compensate for this reduced communication, the numerical scheme takes more iterations

to converge to the same solution. Upto period 2, the effect of lesser time due to communication-avoiding iterations dominates the effect of increased iterations. At higher periods beyond 2, the effect of increased number of iterations become dominant, resulting in an overall increasing trend in time.

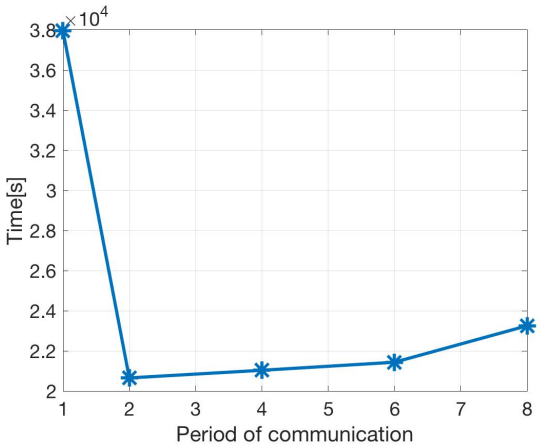

Fig. 9. Reduction in simulation time for state-independent periodic local communication. The global communication also happens with the same period.

D. Scaling

Strong scaling time and speedup plots across various periods of local and global communication are shown for this scheme in Fig 10 and Fig 11 respectively.

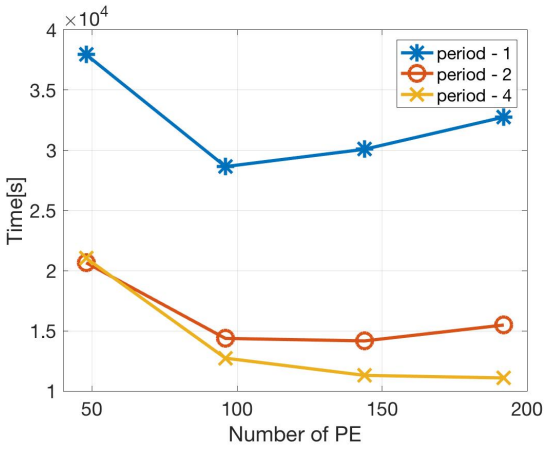

Fig. 10. Strong Scaling of state-independent communication. The simulations are run on $\{48, 96, 144, 192\}$ PEs.

From Fig. 11, we see that the strong scaling speedup performance improves with increase in period. To explain this, consider the case where there is no local communication at all between the PEs (period = ∞) - the speedup for this scenario would be ideal due to no overhead of communication, even though the numerical solver would not converge correctly. Now, as the period increases, communication keeps on decreasing. So the corresponding speedup plot starts approaching the ideal scaling line in an asymptotic sense.

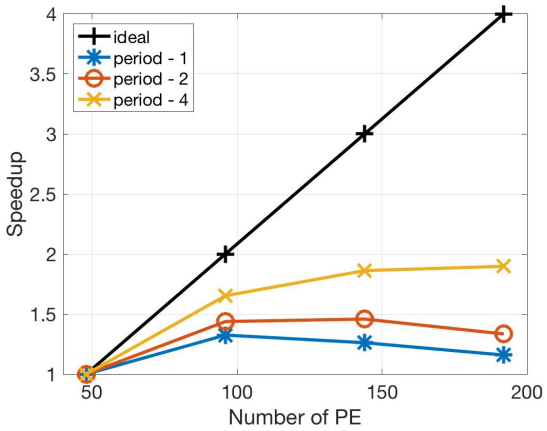

Fig. 11. Speedup of Strong Scaling results in Fig 10. Since we run our simulations in multiples of 48 PEs from 48 to 192, the ideal scaling ranges from 1 to 4.

VI. CONCLUSIONS AND FUTURE WORK

In this work, we describe event-triggered communication methods to speed up simulations in parallel computing environments. We show that state-dependent event-triggered communication methods reduce time due to communication by communicating only when necessary. Strong scaling shows the effectiveness of such methods.

In our future work, we are planning to make significant improvements to the state-dependent event-triggered communication. Firstly, we can design a way to make the threshold for communication adaptive to the change in the boundary values. Secondly, we can also extrapolate the ghost cell values in the receiving PE between instances of communication instead of maintaining them at a constant value until the next communication event happens. Finally, we can remove the global communication for convergence criterion, and communicate the residual information locally to the neighbors.

VII. ACKNOWLEDGEMENT

This research was supported in part by the Notre Dame Center for Research Computing through its computing resources. The work of the first and third authors was supported in part by NSF grants NSF CNS-1544724 and NSF CNS-1739295, and ARO grant Army W911NF-17-1-0072.

REFERENCES

[1] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[2] Daniel Chazan and Willard Miranker. Chaotic relaxation. *Linear algebra and its applications*, 2(2):199–222, 1969.

[3] Gérard M Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM (JACM)*, 25(2):226–244, 1978.

[4] Rafael Bru, Ludwig Elsner, and Michael Neumann. Models of parallel chaotic iteration methods. *Linear Algebra and its Applications*, 103:175–192, 1988.

[5] Dganit Amitai, Amir Averbuch, Moshe Israeli, Samuel Itzikowitz, and Eli Turkel. A survey of asynchronous finite-difference methods for parabolic pdes on multiprocessors. *Appl. Numer. Math*, 12:27–45, 1993.

[6] Andreas Frommer and Daniel B Szyld. On asynchronous iterations. *Journal of computational and applied mathematics*, 123(1-2):201–216, 2000.

[7] Peter K Jimack and Mark A Walkley. Asynchronous parallel solvers for linear systems arising in computational engineering. *Computational technology reviews*, 3:1–20, 2011.

[8] Diego A Donzis and Konduri Aditya. Asynchronous finite-difference schemes for partial differential equations. *Journal of Computational Physics*, 274:370–392, 2014.

[9] Konduri Aditya and Diego A Donzis. High-order asynchrony-tolerant finite difference schemes for partial differential equations. *Journal of Computational Physics*, 350:550–572, 2017.

[10] John Vanrosendale. Minimizing inner product data dependencies in conjugate gradient iteration. 1983.

[11] AT Chronopoulos and Charles William Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25(2):153–168, 1989.

[12] Sivan Avraham Toledo. *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms*. PhD thesis, Massachusetts Institute of Technology, 1995.

[13] Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, UC Berkeley, 2010.

[14] Erin Claire Carson. *Communication-avoiding Krylov subspace methods in theory and practice*. PhD thesis, UC Berkeley, 2015.

[15] Ichitaro Yamazaki, Hartwig Anzt, Stanimire Tomov, Mark Hoemmen, and Jack Dongarra. Improving the performance of ca-gmres on multicores with multiple gpus. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 382–391. IEEE, 2014.

[16] Protonu Basu, Anand Venkat, Mary Hall, Samuel Williams, Brian Van Straalen, and Leonid Oliker. Compiler generation and autotuning of communication-avoiding operators for geometric multigrid. In *2013 20th International Conference on High Performance Computing (HiPC)*, pages 452–461. IEEE, 2013.

[17] Marc Baboulin, Simplice Donfack, Jack Dongarra, Laura Grigori, Adrien Rémy, and Stanimire Tomov. A class of communication-avoiding algorithms for solving general dense linear systems on cpu/gpu parallel machines. *Procedia Computer Science*, 9:17–26, 2012.

[18] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for strassen's matrix multiplication. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 193–204. ACM, 2012.

[19] Benjamin Lipshitz, Grey Ballard, James Demmel, and Oded Schwartz. Communication-avoiding parallel strassen: Implementation and performance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 101. IEEE Computer Society Press, 2012.

[20] Karthik Murthy and John Mellor-Crummey. Communication avoiding algorithms: Analysis and code generation for parallel systems. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 150–162. IEEE, 2015.

[21] WPMH Heemels, Karl Henrik Johansson, and Paulo Tabuada. An introduction to event-triggered and self-triggered control. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 3270–3285. IEEE, 2012.

[22] Manuel Mazo and Paulo Tabuada. Decentralized event-triggered control over wireless sensor/actuator networks. *IEEE Transactions on Automatic Control*, 56(10):2456–2461, 2011.

[23] James Dinan, Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. An implementation and evaluation of the mpi 3.0 one-sided communication interface. *Concurrency and Computation: Practice and Experience*, 28(17):4385–4404, 2016.

[24] Soumyadip Ghosh, Jiacai Lu, Vijay Gupta, and Gretar Tryggvason. Fast parallel computation using periodic synchronization. In *2018 Annual American Control Conference (ACC)*, pages 1659–1664. IEEE, 2018.