Coalesced communication: a design pattern for complex parallel scientific software

Hywel B. Carver[a,b], Derek Groen[b], James Hetherington[b], Rupert W. Nash[b], Miguel O. Bernabeu[b,a], Peter V. Coveney[b]

[a]*CoMPLEX, University College London, London, United Kingdom*
[b]*Centre for Computational Science, University College London, London, United Kingdom*

Abstract

We present a new design pattern for high-performance parallel scientific software, named coalesced communication. This pattern allows for a structured way to improve the communication performance through coalescence of multiple communication needs using two communication management components. We apply the design pattern to several simulations of a lattice-Boltzmann blood flow solver with streaming visualisation which engenders a reduction in the communication overhead of approximately 40%.

Keywords: parallel computing, parallel programming, high-performance computing, message passing

1. Introduction

High-performance parallel scientific software often consists of complex, multi-functional, multi-physics software components, run on infrastructures which are increasingly large and frequently hybrid in nature (e.g., featuring many-core architectures or distributed systems). Orchestrating the work of these components requires advanced software engineering and design approaches to manage the attendant complexity. The result is that the structure of high-performance computing codes is moving towards the use of higher-level design abstractions. One way to capture these design abstractions is through the definition of *design patterns*. Design patterns are com-

---

[1]E-mail: hywel.carver.09@ucl.ac.uk (Hywel B. Carver), p.v.coveney@ucl.ac.uk (Peter V. Coveney)

monly applied in software engineering [1]. They are formal definitions which describe a specific solution to a design problem, and can be found in a range of scientific and engineering disciplines. With high performance computing (HPC) codes growing in complexity, existing design patterns are more commonly applied in HPC and numerous new design patterns have emerged [2, 3].

Here we present a new design pattern: coalesced communication. In this pattern, each component registers the communication tasks it will require during the different stages, or *steps*, of execution with a central registry. We refer to each component which wishes to register communication requests as a *Client*. This registry analyses the required communications and combines requests from each Client at appropriate steps of the execution. This allows work of one Client (such as a scientific kernel) to overlap with the communication of another Client (such as streaming visualisation or error correction), and results in a single synchronization point between processes during each step.

Several groups have experimented with the coalescence of communication, although none of these have developed this into a generalised design pattern. *Bae et al.* [4] benchmark the coalescence of communication as a factor influencing code complexity and efficiency within two algorithms. *Bell et al.* [5] investigate the performance benefit of overlapping communication with communication, which is an alternative method to reduce the number of synchronisation points. *Chavarria et al.* [6] implement a form of coalescence in a High-Performance Fortran compiler for situations where one code location has multiple communication events, and find a reduction of up to 55% in communication volume. *Chen et al.* [7] find similar performance improvement when applying coalescing in programs written in Unified Parallel C, and *Koop et al.* [8] report significant improvements in throughput when using low-level coalescence for sending small MPI messages.

2. Coalesced communication

The coalesced communication pattern is applicable to any parallel software which carries out multiple tasks, and therefore has a range of communication needs. These communication needs may, for example, include exchanges required for one or more scientific kernels, visualisation, steering, dynamic domain decomposition, coupling with one or more external programs, introspection or error recovery. Of course, each of these Clients could do its own communication internally, but this can be highly inefficient

from a performance perspective due to the large number of synchronisation points with other processes. The coalesced communication pattern allows us to improve the communication performance by reducing the number of synchronisation points in an organised way.

Within the coalesced communication pattern, each Client registers with an administrative object called the *StepManager*, and all communication is indirected through a central store of communication requirements called the *CommunicationsManager* object. The relations of these objects are shown in Figure 1. In each of several *Steps*, a call back is made to each Client to carry out those computations that are safe to perform during that step, while the CommunicationsManager object makes the appropriate MPI calls to initiate non-blocking message passing for each requested piece of communications. In this way, the communications of all Clients can be overlapped with their calculation, potentially providing substantial performance gains. In addition, the bundling of all the non-blocking communications reduces the number of synchonisation points here to one.

We present the sequence of events for an application with two Clients in Figure 2. Here we see computation callbacks preceding and following each of the MPI send, receive, and wait calls. For example, computation callbacks are made to each Client after the CommunicationsManager makes the MPI send calls, while it waits to receive the incoming data. The incoming data are placed into buffers registered with the CommunicationsManager at the beginning of each step, but the data is only safe to use following completion of the Wait call made by CommunicationsManager.

3. Implementation

We have implemented the coalesced communication design pattern within the HemeLB lattice-Boltzmann simulation environment, which is intended to accurately model cerebrovascular blood flow. HemeLB is written in C++ and aims to provide timely and clinically relevant assistance to neurosurgeons [9]. HemeLB contains a range of functionalities, including the core lattice-Boltzmann kernel, visualisation modules and a steering component which allows for interactive use of the application. HemeLB has been shown to efficiently model sparse geometries using up to at least 32,768 compute cores [10]; *inter alia*, has been used for a variety of scenarios [11, 9].

The primary Clients registered with the StepManager within HemeLB are those raised by the core lattice-Boltzmann kernel, an *in situ* visualisa-
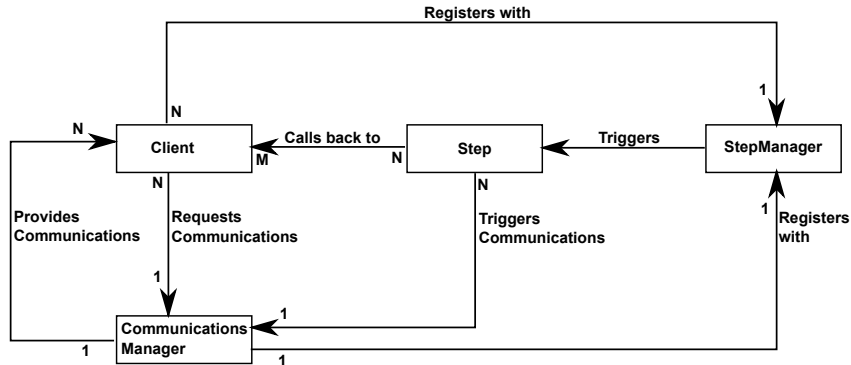
Figure 1: Entity relationship diagram of the coalesced communication design pattern.

tion module and an module for introspective monitoring. However, HemeLB will frequently run with additional Clients as there are a number of optional modules, such as the computational steering server. Within this article we focus on only the core lattice-Boltzmann communications and the visualisation communications.

4. Performance Tests

We have run HemeLB on 1024 cores on the HECToR Cray XE6 machine in Edinburgh, United Kingdom, using a sparse cerebrovascular bifurcation simulation domain which contains 19,808,107 fluid sites. Our simulations run for 2000 steps with three different settings, rendering respectively 10, 100 and 200 images using the visualisation module. We repeated each run both with and without coalesced communication enabled, using a compile-time parameter to toggle this functionality. We measured the total time spent on the simulation, on all communications, and on local operations required for constructing the images.

We present the results of our performance tests in Table 1. Based on our measurements we find that the communication overhead in our coalesced runs amounts to between 57 and 63% of the overhead in the non-coalesced runs. When we render more images per timestep, the absolute performance benefit increases while relative performance benefit slightly decreases. However, the frame rate we obtain for the runs with 200 images generated is already sufficient for real-time visual inspection of the data. The time spent on vi-
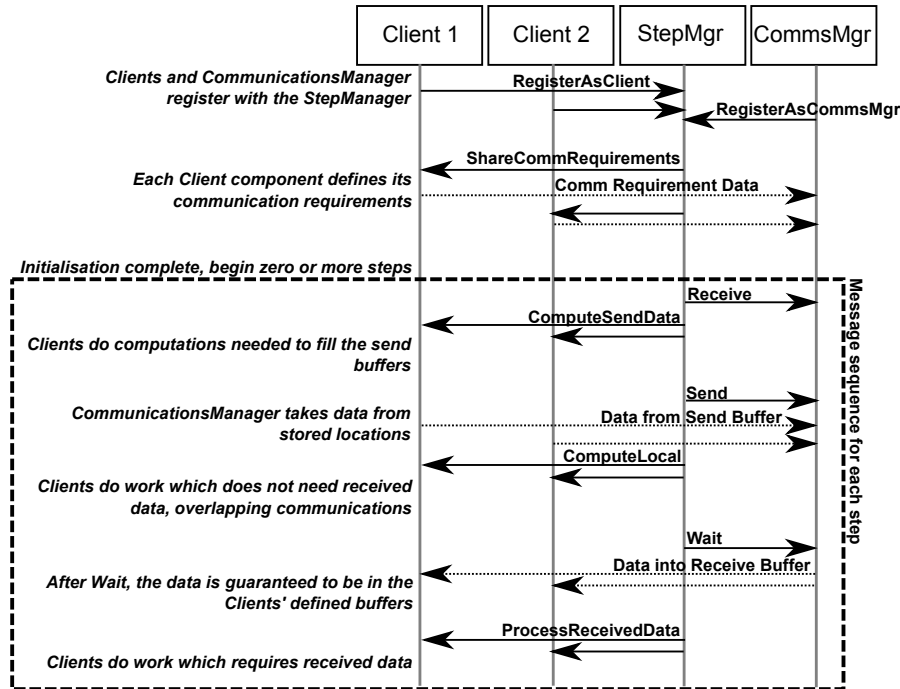
4

Figure 2: Message sequence chart of the coalesced communication pattern, generalized for an application with two Client components which require communications. Function calls and data movements are indicated respectively with solid and dashed arrows. The Step-Manager and CommunicationsManager objects are abbreviated respectively as StepMgr and CommsMgr. Time proceeds vertically downwards.

sualisation is 0.0034 second per image, and scales linearly with the number of images rendered.

5. Discussion and conclusions

We have presented the coalesced communication design pattern, which allows the coalescence of the interprocess communications of multiple Client components within complex parallel scientific software. We have demonstrated the benefit of adopting the design pattern based on an implementation in a blood flow application. Here the use of coalesced communication reduces the total communication overhead of the simulations, which have two primary Clients, by approximately 40%. This improvement results in the application taking about 7% less time overall, making it more responsive when

Table 1: Performance results of our HemeLB simulations, run with and without the coalesced communication strategy. Each simulation ran for 2000 time steps, using 1024 cores and modelling blood flow in a bifurcation simulation domain. We ran our simulations rendering respectively 10 images (first two rows), 100 images (middle two rows), and 200 images (last two rows) at evenly spaced time intervals during execution.

# of images	Coalesced Comm.	Total time [s]	Comm. time [s]	Vis. time [s]
10	enabled	27.6	2.36	0.03
10	disabled	29.3	4.07	0.03
100	enabled	30.0	3.13	0.34
100	disabled	31.9	5.15	0.33
200	enabled	32.7	3.82	0.68
200	disabled	34.8	6.07	0.66

applied for clinical or scientific purposes. The design pattern can be directly applied in other parallel scientific software projects, allowing for a structured way to improve the communication performance through coalescence.

6. Acknowledgements

References

[1] G. Erich, H. Richard, J. Ralph, V. John, Design patterns: elements of reusable object-oriented software, Addison Wesley Publishing Company, Reading, United Kingdom, 1995.

[2] J. L. Ortega-Arjona, Patterns for Parallel Software Design, John Wiley and Sons Ltd., Chichester, United Kingdom, 2010.

[3] T. Mattson, B. Sanders, B. Massingill, Patterns for parallel programming, 1st Edition, Addison-Wesley Professional, 2004.

[4] S. Bae, S. Ranka, A comparison of different message-passing paradigms for the parallelization of two irregular applications, The Journal of Supercomputing 10 (1) (1996) 55–85.

[5] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, K. Yelick, An evaluation of current high-performance networks, in: Parallel and Distributed Processing Symposium, 2003, 2003, p. 10 pp.

[6] D. Chavarría-Miranda, J. Mellor-Crummey, Effective communication coalescing for data-parallel applications, in: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, ACM, New York, NY, USA, 2005, pp. 14–25.

[7] W.-y. Chen, C. Iancu, K. Yelick, Communication optimizations for fine-grained UPC applications, in: In Proceedings of the International Conference on Parallel Architecture and Compilation Techniques, 2005, pp. 267–278.

[8] M. J. Koop, T. Jones, D. K. Panda, Reducing connection memory requirements of MPI for infiniband clusters: A message coalescing approach, Cluster Computing and the Grid, IEEE International Symposium on (2007) 495–504.

[9] M. D. Mazzeo, P. V. Coveney, HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries, Computer Physics Communications 178 (12) (2008) 894–914. `doi:10.1016/j.cpc.2008.02.013`.

[10] D. Groen, J. Hetherington, H. B. Carver, R. W. Nash, M. O. Bernabeu, P. V. Coveney, Analyzing and Modeling the Performance of the HemeLB Lattice-Boltzmann Simulation Environment, submitted to the Journal of Computational Science`arXiv:1209.3972`.

[11] H. B. Carver, R. W. Nash, M. Bernabeu, J. Hetherington, D. Groen, T. Krueger, P. V. Coveney, Choice of boundary condition and collision operator for lattice-Boltzmann simulation of intermediate Reynolds number flow in complex domains, submitted to Phys Rev. E.