A High Performance C++ Generic Benchmark for Computational Epidemiology

Aniket Pugaonkar
Virginia Bioinformatics Institute &
Dept. of Compter Science,
Virginia Tech, USA
Email: aniketnp@vbi.vt.edu

Keith R. Bisset
Virginia Bioinformatics Institute,
Virginia Tech, USA
Email: kbisset@vbi.vt.edu

Sandeep Gupta
Virginia Bioinformatics Institute,
Virginia Tech, USA
Email: sandeep@vbi.vt.edu

Madhav V. Marathe
Virginia Bioinformatics Institute &
Dept. of Compter Science,
Virginia Tech, USA
Email: mmarathe@vbi.vt.edu

***Abstract*—An effective tool used by planners and policy makers in public health, such as Center for Disease Control (CDC), to curtail spread of infectious diseases over a given population is contagion diffusion simulations. These simulations model the relevant characteristics of the population (age, gender, income etc.) and the disease (attack rate, etc.) and compute the spread under various configuration and plausible intervention strategies (such as vaccinations, school closure, etc.). Hence, the model and the computation form a complex agent based system and are highly compute and resource intensive.**

In this work, we design a benchmark consisting of several kernels which capture the essential compute, communication, and data access patterns for such applications. For each kernel, the benchmark provides different evaluation strategies. The goal is to (a) derive alternative implementations for computing the contagion by combining different implementation of the kernels, and (b) evaluate which combination of implementation, runtime, and hardware is most effective in running large scale contagion diffusion simulations. Our proposed benchmark is designed using C++ generic programming primitives and lifting sequential strategies for parallel computations. Together, these lead to a succinct description of the benchmark and significant code reuse when deriving strategies for new hardware. For the benchmark to be effective, this aspect is crucial, because the potential combination of hardware and runtime are growing rapidly thereby making infeasible to write optimized strategy for the complete contagion diffusion from ground up for each compute system.

I. Introduction

By far, the High Performance Linpack (HPL) or Top 500 benchmark [10] is the most widely recognized and discussed metric for ranking high performance computing system. However, with the new trends in hardware, we need take into account the interconnect and data access patterns of the underlying application as well [8]. Therefore, there arises a need for application specific benchmarking to capture the essential compute, communication and data access patterns of high performance applications.

Interaction based simulations provide a realistic insight on disease spreading mechanisms which are modeled on social contact networks and daily activities of people in the network. Such models rely on high performance computing systems to provide fast and accurate results for making quick and sensible decisions during an outbreak. Current large scale simulation for contagion diffusion exists for Blue Waters machine using Charm++ runtime [12]. In this work, we present an application specific C++ benchmark to evaluate such systems using a suite of six generic kernels with different implementation strategies to determine which combination of the implementation, runtime, and hardware is most effective in simulating contagion over large social contact networks.

II. Contributions

We developed specifications and epidemiology oriented metrics for our kernels and provide their generic implementations in C++. The kernels we provide for agent based and contact based contagion models are designed to capture the computational complexity (and not semantics) of algorithms [1],[2] which are widely used in tools which simulate contagion-diffusion over social network. And lastly, we develop scalable shared and distributed memory generic implementation of kernels using task based parallelism and message passing interface.

III. Related Work

Application specific benchmarking is essential for two reasons (a) to find better correlation between an application and the machine running it and (b) to help choose most appropriate hardware-software configuration for given application and system parameters. This motivated the Graph 500 [3] benchmark to be developed for applications with graphs as their core analytical workloads. The Boost Graph Library [4] is based on BOOST C++ framework and generic programming principles. PBGL [6] is distributed graph library built by lifting, i.e, providing distributed implementation for various interfaces and operators in BGL.

IV. High Level Specs and Interface Design

A. Specifications

a. **`kernel0<DegreeDistOfPersonsT, DegreeDistOfLocationsT>`**: Takes as input, the degree distribution for persons and locations. The output is list of person and location pairs. Each pair (p, l) represents an activity by person p at location l.

b. **`kernel1<GraphT,DiseaseModelT<ModelT, CriteriaT>>`**: Produces an activity graph defined by the (p, l) activity list. The parameters `GraphT` and `DiseaseModelT` control the type of the generated network graph and the initial state of the vertices respectively.

c. **`kernel2<GraphT,DiseaseModelT<ModelT, CriteriaT>>`**: Produces a contact graph.

d. **`kernel3<LocationGroupT>`**: Distributes the locations in activity graph into location groups. The data structures for location group are controlled through the `LocationGroupT` parameter.

e. **`kernel4<GraphT,DiseaseModelT<ModelT, CriteriaT>,InterventionT, ContagionFunctorT>`**: Simulates contagion over the activity graph.

f. **`kernel5<GraphT,DiseaseModelT<ModelT, CriteriaT>,InterventionT, ContagionFunctorT>`**: Simulates contagion over the contact graph.

B. Interfaces

The implementations for the parameters much conform to the constraint in order to derive a functional benchmark.

a. **GraphT**:

i. Must provide iterator-style access to vertices and edges. We adapt our interface to common vertex and edge iterators defined in `Boost.Graph`

ii. Must have provision for property map to associate vertices and edges with their respective properties.

iii. In non-mutable graph, where vertices/edges cannot be deleted from memory, logical deletion (such as filtering) can be used.

Note: We develop similar interfaces for other benchmark parameters such as `DiseaseModelT`, `InterventionT` and `ContagionFunctorT`. However, the constraints are not provided here due to lack of space.

V. Lifting Sequential methods for Shared and Distributed memory algorithm

We develop a task based work stealing model to lift the sequential algorithms for shared and distributed memory algorithms. A worker is either a thread (shared memory) or an MPI process (distributed). Our approach is based upon Intel TBB [7] task based scheduling. An idle worker may steal the task from task pool. An MPI worker can be assigned a task from a task pool which it can further spawn threads (workers, again) to create additional sub-tasks.

VI. Performance Evaluation and Metrics

The **Total Interactions Per Second** (TIPS) metric provides us a number for particular combination of strategy, runtime and hardware combination. The number of interactions in a contact network are independent of disease parameters where as a weaker metric such as *number of infections per second* depends upon disease parameters such as infectivity, transmissibility etc [5]. We evaluate or benchmark on two of our systems - BlueRidge (ARC, Virginia Tech) and Shadowfax (VBI, Virginia Tech). Preliminary results indicate that the BlueRidge system is more scalable than Shadowfax.

VII. Conclusion and Ongoing Work

Our work is lies on the intersection of generic programming and high performance computing for computational epidemiology. We present a suite of kernels that together form benchmark for contagion diffusion simulation. We provide an encoding of the benchmark specifications using C++11 templates and iterators that is generic and composable, i.e, different implementations of kernels can be composed together to arrive at alternative implementations of the benchmark.

Ongoing work: Our current work is focused upon two major aspects (a) Standardization – developing codes for our benchmark so to make it compatible to any standard graph library which implements its basic specifications, and (b) Distributed/shared memory implementation – we aim to lift the sequential implementation for large scale shared memory and distributed memory implementations without affecting the genericness and simplicity of the benchmark.

References

[1] Barrett, Christopher L., et al. "EpiSimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks." Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008.

[2] Bisset, Keith R., et al. "EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems." Proceedings of the 23rd international conference on Supercomputing. ACM, 2009.

[3] Murphy, Richard C., et al. "Introducing the graph 500." Cray Users Group (CUG) (2010).

[4] Siek, Jeremy G., et al. Boost Graph Library: User Guide and Reference Manual, The. Pearson Education, 2001.

[5] N. Bailey. The Mathematical Theory of Infectious Diseases and Its Applications. Hafner Press, New York, 1975.

[6] Gregor, Douglas, and Andrew Lumsdaine. "The Parallel BGL: A generic library for distributed graph computations." Parallel Object-Oriented Scientific Computing (POOSC) (2005): 2.

[7] Chuck Pheatt. 2008. Intel threading building blocks. J. Comput. Sci. Coll. 23, 4 (April 2008), 298-298.

[8] M. Heroux and J. Dongarra. Towards a New Metric for Ranking High Performance Computing Systems, UTK EECS and Sandia National Labs Report SAND2013-4744, June 2013

[9] Gregor, Douglas, and Andrew Lumsdaine. "Lifting sequential graph algorithms for distributed-memory parallel computation." ACM SIGPLAN Notices 40.10 (2005): 423-437.

[10] Dongarra, J., et al. Top 500 Supercomputer Sites. 1999; Available from: http://www.top500.org

[11] Madhav Marathe, Anil Kumar S. Vullikanti. Computational Epidemiology. Communications of the ACM, Vol. 56 No. 7, Pages 88-96 10.1145/2483852.2483871.

[12] Yeom, Jae-Seung, et al. "Overcoming the Scalability Challenges of Epidemic Simulations on Blue Waters." Proceedings of the IEEE International Parallel & Distributed Processing Symposium (to appear). 2014.