Title

Network-Theoretic Classification of Parallel Computation Patterns

Journal

International Journal of High Performance Computing Applications (IJHPCA), 26(2)

Authors

Whalen, Sean
Engle, Sophie
Peisert, Sean
et al.

Publication Date

2012-03-18

Peer reviewed

Network-Theoretic Classification of Parallel Computation Patterns

Sean Whalen, Sophie Engle, Sean Peisert, Matt Bishop

December 14, 2012

Abstract

Parallel computation in a high performance computing environment can be characterized by the distributed memory access patterns of the underlying algorithm. During execution, networks of compute nodes exchange messages that indirectly exhibit these access patterns. Identifying the algorithm underlying these observable messages is the problem of latent class analysis over information flows in a computational network. Towards this end, our work applies methods from graph and network theory to classify parallel computations solely from network communication patterns. Pattern classification has applications to several areas including anomaly detection, performance analysis, and automated algorithm replacement. We discuss the difficulties encountered by previous efforts, introduce two new approximate matching techniques, and compare these approaches using massive datasets collected at Lawrence Berkeley National Laboratory.

1 Introduction

The field of High performance computing (HPC) is undergoing dramatic change as researchers plan for next-generation exascale systems, and distributed computing in general has seen rapid growth due to the cloud. Understanding the characteristics of distributed computation is essential for improving the efficiency and scalability of software in these environments [1] as well as growing concerns about security. Static analysis of binaries or source is one (costly) approach; another is to examine computations indirectly by observing their patterns of communication.

Our work uses the characteristics of runtime communication patterns to infer what code or algorithm is actually executing in a distributed environment. Towards this end, we build classifiers

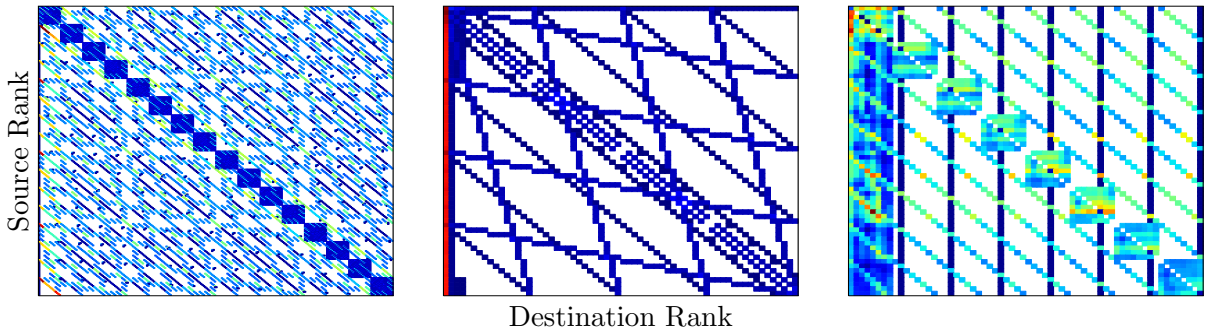


Figure 1: Adjacency matrices for individual runs of performance benchmark MADBENCH (256 nodes), atmospheric dynamics simulator FVCAM (64 nodes), and linear equation solver SUPERLU (64 nodes). The number of bytes sent between ranks is linearly mapped from dark blue (lowest) to red (highest), with white indicating an absence of communication.

using a combination of methods from graph theory, network theory, and hypothesis testing. Classification in this setting is the problem of structural pattern recognition applied to unknown parallel computations that reveal themselves indirectly via messages exchanged by a network of compute nodes. Identifying the underlying algorithm is thus a type of latent class analysis where a "hidden" algorithm must be identified only from observable information flows. This task is non-trivial: compilers, architectures, libraries, datasets, parameters, and software flaws can each potentially influence communication patterns. Additionally, different algorithms can express similar patterns, and different implementations of the same algorithm can express different patterns.

In this paper we first describe how communication patterns are dynamically captured from running applications and the relation of these patterns to abstract computational classes called *dwarfs*. Methods from graph and network theory are introduced and their shortcomings discussed. We then demonstrate multiple approaches to achieve efficient, approximate matching of communication patterns that avoid the computational costs of static analysis.

2 Background

2.1 Communication Logging

Message Passing Interface (MPI) is a communications protocol standard used by many parallel programs to exchange data using a distributed memory model. There are several implementations such as OpenMPI and MPICH, each based on the idea of logical processors with unique labels

called *ranks* placed in groups called *communicators*. MPI programs have an initialization phase where each processor joins a communicator and is assigned a rank, and a finalization phase to gracefully terminate after computation.

The *Integrated Performance Monitoring* (IPM) library [2] provides low-overhead performance and resource profiling for parallel programs. It logs features of MPI calls such as the call name, the source and destination rank, the number of bytes sent, and aggregate performance counters such as the number of integer and floating point operations. The library is enabled either at compile time or runtime and uses library interposition to intercept MPI calls at runtime.

Consider the following abbreviated IPM log entry:

<hent call="MPI_Isend" bytes="599136" orank="1" count="26" />

These entries become rows in a two dimensional feature matrix where rows are individual calls and columns are call features. Call names are mapped to unique integers so the contents of the feature matrix are purely numerical. The above entry then becomes:

$$\begin{pmatrix} \text{int(MPI\_Isend)} & 599136 & 1 & 26 \end{pmatrix}$$

The result is a matrix of features for each run of a parallel program. By varying datasets, parameters, the number of compute nodes, and other factors, we obtain multiple matrices for each program. These matrices are then converted to directed graphs by treating ranks as nodes and calls between ranks as edges. The task at hand, then, is to apply structural and statistical pattern analysis to differentiate patterns of parallel computation.

2.2 Computational Dwarfs

A *computational dwarf* is "a pattern of communication and computation common across a set of applications" [3]. Each dwarf is an equivalence class of computation independent of the programming language or numerical methods used for a particular implementation. The common use of shared libraries such as BLAS and LAPACK provides some evidence of these equivalence classes, though dwarfs imply a level of algorithmic equivalence beyond code reuse.

Colella et al. identified seven dwarfs in HPC applications [4]: dense linear algebra, sparse linear algebra, spectral methods, n-body methods, structured grids, unstructured grids, and monte

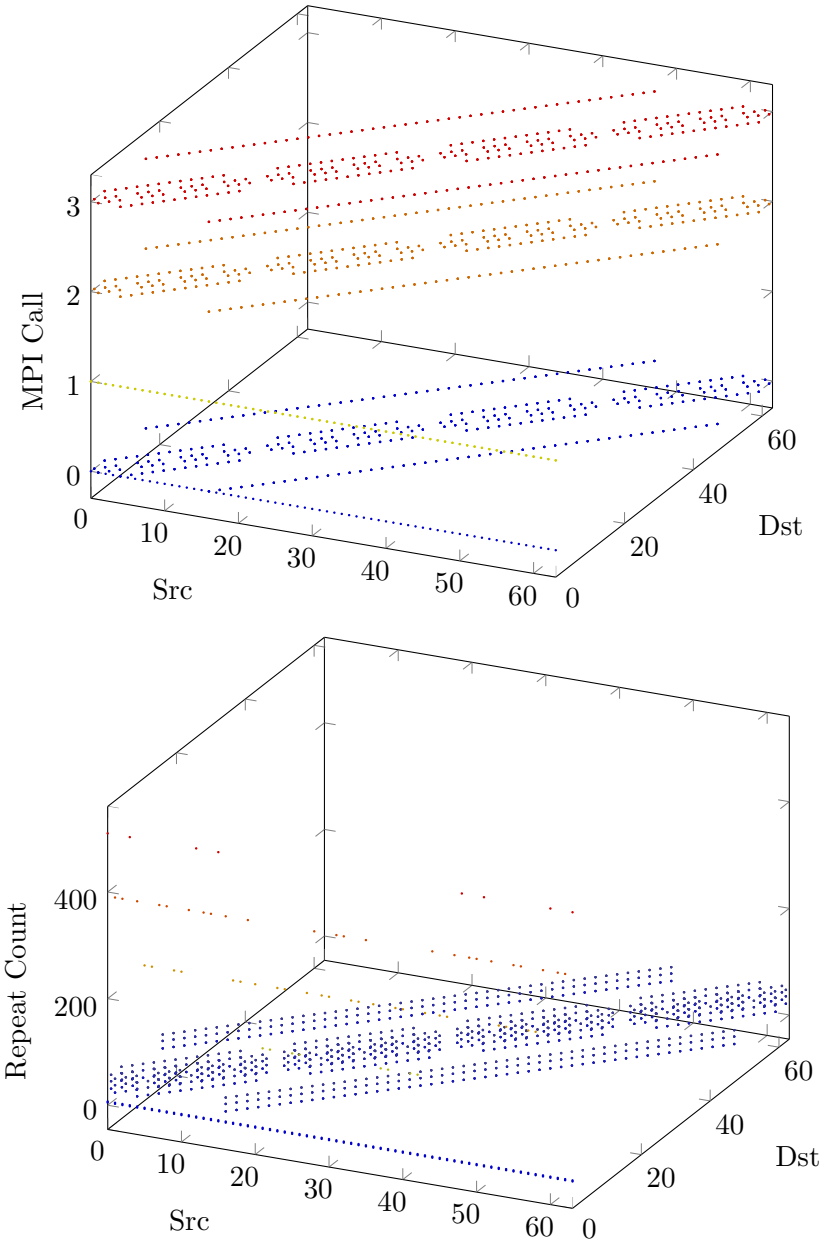


Figure 2: Adjacency matrix of general relativity simulator CACTUS augmented by MPI call (top) and message size (bottom). For purposes of display, MPI call names are mapped to integers; identical MPI calls receive the same color. In the bottom figure, color is a linear mapping of the number of repeats from dark blue (lowest) to red (highest). Such plots show the structure seen in adjacency matrices extends to features other than the source and destination ranks of MPI communications.

carlo methods. Asanovich et al. asked if these seven also captured patterns from areas outside of HPC [3]. They found six additional dwarfs were needed to capture the distinct patterns of computation outside HPC including combinational logic, graph traversal, dynamic programming, backtrack and branch/bound, graphical models, and finite state machines. Originally named in reference to the seven dwarfs of Snow White, they are now commonly called *computational motifs* due to these additional patterns. However, we use the original term to prevent confusion with the network motifs presented in this paper.

Distributed memory parallel programs, then, will fall into one or more of these 13 dwarf classes. If the variance of the expressed patterns is bounded, identification of the dwarf class should be possible solely from observed communications.

2.3 Visualization

Consider a three node communicator where rank 0 sends messages to ranks 1 and 2, rank 1 sends a message to rank 2, and ranks 1 and 2 send messages back to 0. These messages have the following adjacency matrix representation:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Adjacency matrices are commonly visualized as a grid where the axes are rank numbers and filled pixels denote ranks that exchanged one or more messages. Different communication features such as the number of messages exchanged or their total size can be stored in the matrix and color-mapped to provide additional insight. Such visualizations are commonly used to examine communication patterns and have been offered as evidence for the existence of computational dwarves.

The adjacency matrices for single runs of three different parallel programs are shown in Figure 1. Another layer of structure can be seen by extending the adjacency matrix into a third dimension, mapping an additional communication feature onto the z-axis. Figure 2 shows such mappings for MPI messages and their counts using general relativity simulator CACTUS.

Communication patterns are strongly tied to distributed memory access within a parallel pro-

gram. To see this, examine the diagonal of Figure 1's center panel and note the communication between a rank and its immediate neighbors. Such a pattern is generated by finite difference equations and is found across many HPC applications. Another type of equation will have a different visual signature unless its pattern of distributed memory access is similar.

The structure seen in Figures 1 and 2 is typical of the applications we examined and suggests that classification is possible. By the same argument, however, distinguishing applications within the same dwarf class may be difficult due to their topological similarity. Complicating matters, the same program may alter its communications given different parameter values, datasets, or communicator sizes (see Figure 3). As a result, we cannot simply compare adjacency matrices to classify the underlying computation.

3 Classification

This section presents both structural (topological) and statistical approaches to recognizing patterns in communication graphs constructed from IPM logs. These patterns are used to predict the unknown computation underlying the observed communications; this is the process of classification. We first introduce several approaches that failed to accurately classify our datasets. Their shortcomings motivate our later approaches.

Communications are represented as directed graphs with ranks as nodes and MPI calls as edges. Call names are stored as edge labels (colors) and multiple edges exist between nodes if more than one type of MPI message is exchanged. Graphs with labeled nodes and/or edges are called *attributed*

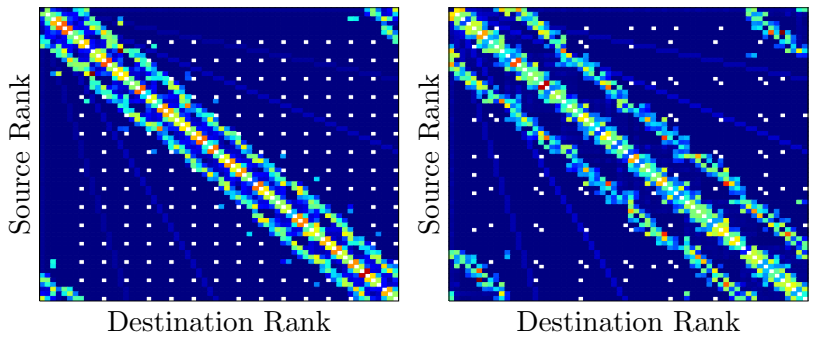
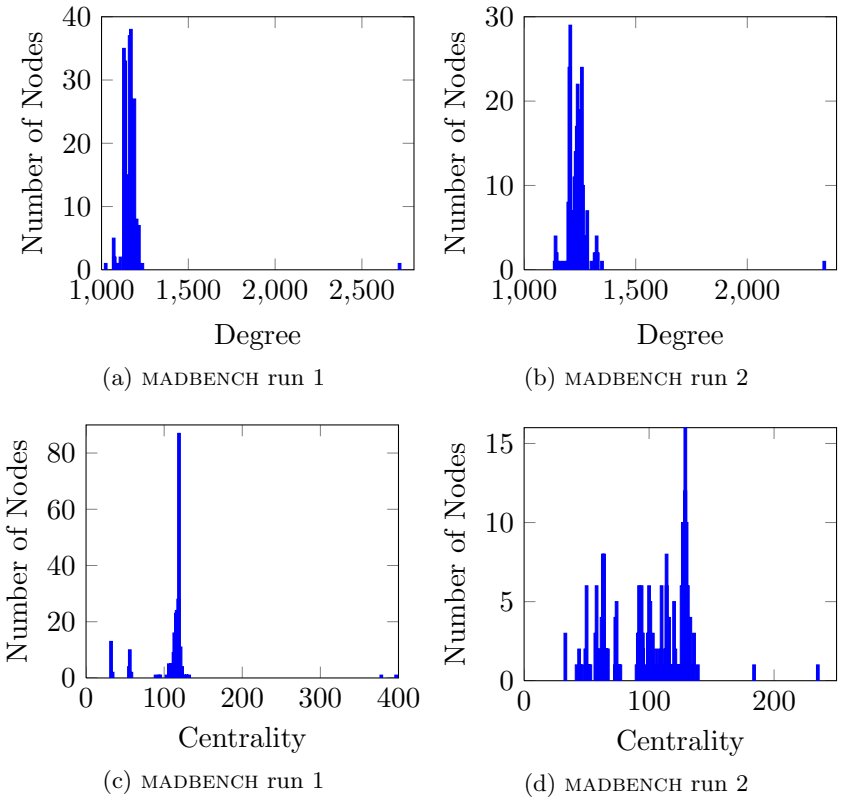


Figure 3: Data dependent topology demonstrated by molecular dynamics simulator NAMD under different molecular arrangements. The number of bytes sent between ranks is linearly mapped from dark blue (lowest) to red (highest), with white indicating an absence of communication.



(a) MADBENCH run 1 (b) MADBENCH run 2

(c) MADBENCH run 1 (d) MADBENCH run 2

Figure 4: Node degree and betweenness centrality distribution for two runs of performance benchmark MADBENCH. These runs exhibit similar degree distributions but different centrality distributions. Classification using these statistics is error prone due to their variance over different runs of the same program.

relational graphs (ARGs) [5] and this data reduces the search space of some algorithms such as the subgraph isomorphism test introduced in Section 3.2. The full details of our datasets and evaluation are presented in Section 3.6.

3.1 Node Distributions

The first statistical graph measure, the *node degree distribution*, counts the total number of nodes having a particular number of edges (*degree*). This analysis is restricted to the out-degree distribution measuring only outbound edges. For example, the adjacency matrix in Section 2.3 has two nodes of degree 2 and a single node of degree 1. The node degree distribution for two individual runs of the MADBENCH performance benchmark are shown in Figures 4a and 4b.

Node degree distributions are a summary statistic over the adjacency matrix and the types of messages exchanged, reflecting the layered per-call adjacency matrices in the left panel of Figure 2.

Though offering additional insight, they summarize a single aspect of the graph that may fail to distinguish different patterns. In these cases, the notion of *centrality* can be helpful.

Centrality measures the importance of a node in the graph, and this importance can be defined in several ways. We examine the *betweenness centrality* (C_B), measuring the percent of shortest paths passing through a node v in an undirected graph [6]:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where $\sigma_{st}(v)$ is the number of shortest paths between nodes s and t that pass through v. This number is normalized by the total number of shortest paths between s and t. The C_B of v, then, is the sum of these normalized shortest path counts over all node pairs not containing v.

Intuitively, nodes acting as coordinators of computation such as rank 0 have high C_B. As an artifact of IPM, broadcast messages also have high centrality. This can be seen in Figures 4c and 4d. Note that despite similar degree distributions, the second run has a very different centrality distribution.

In this example, relying solely on centrality to classify the computation results in a false negative (incorrectly predicting the patterns are from different algorithms). The degree distribution works for this example but will result in a false positive (incorrectly predicting patterns are from the same algorithm) for many others. Thus, multiple such measures are important for differentiating parallel computations.

However, these statistics are based solely on topological properties of the computational network and do not incorporate other attributes of information flow. They are also sensitive to the number of compute nodes. It is vital that any classification is independent of the communicator size, whether the computation is performed with 32, 64, 128, or more nodes.

3.2 Graph Isomorphisms

In our setting, two parallel computations using the same algorithm are often *isomorphic*: given the set of vertices $V(G)$ and edges $E(G)$ for some graph G, graph isomorphism is a bijection between

two graphs G and H:

$$f : V(G) \rightarrow V(H)$$

This mapping preserves edge structure: $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$ [7]. Isomorphic communication patterns imply that messages are passed between the same nodes in each graph regardless of the assigned rank number, akin to scrambling the columns of the adjacency matrix.

As two computations performed on different numbers of nodes cannot be isomorphic, we are instead interested in the related problem of *subgraph isomorphism*. Two graphs are subgraph isomorphic if some subgraph of G is isomorphic to H: for example, if some subset of the communication graph for atmospheric dynamics simulator FVCAM with 256 nodes is isomorphic to the same program run with 128 nodes.

Graph isomorphism has not been proven NP-complete or a member of P, while subgraph isomorphism is NP-complete via reduction to the maximum clique problem [8]. Runtime complexity is of serious concern as HPC networks often contain hundreds or thousands of nodes. Our investigations focus on Ullman's subgraph isomorphism algorithm [9], the VF2 algorithm [10], and the NetworkX implementation of VF2 [11]. The Ullman and VF2 algorithms are implemented in the VFLib library [12] and have worst-case time complexity $\mathcal{O}(N!N^2)$ and $\mathcal{O}(N!N)$, respectively.

IPM logs are converted to directed graphs with MPI calls encoded as edge attributes. As discussed earlier, these attributed relational graphs prune the state space of the isomorphism test and reduce false positives [5]. However, the primary problem with isomorphism-based classification is its exact matching requirements that result in false negatives for all data-dependent topologies (see Figure 3). Ideally, such topologies will be classified the same if differences are within some bounded variance. To address this, the next section introduces an approximate matching approach using statistical hypothesis testing.

3.3 Hypothesis Testing

A graph isomorphism test requires exact node correspondence between two graphs. This requirement results in many false negatives when applied to communication patterns whose statistics can vary with architecture, communicator size, parameters, and datasets. Current approximate graph

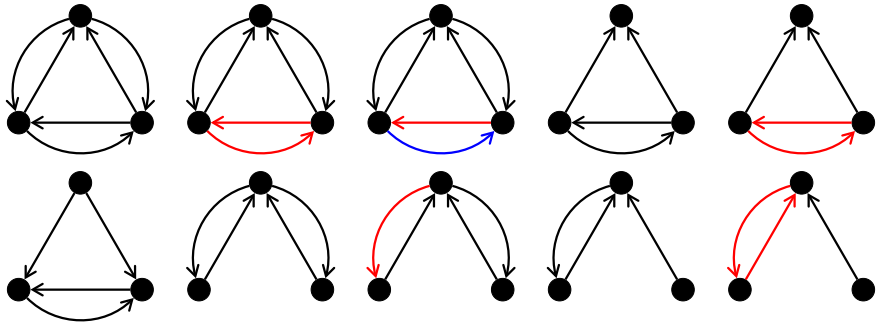


Figure 5: A random sample of common 3-node motifs with different edge colorings. Black represents all calls in single color motifs, while collective (black) and point-to-point (red) calls are represented separately in 2-color motifs. In 3-color motifs, point-to-point calls are further divided into send (red) and receive (blue) calls.

matching methods such as graph edit distance [5] are often more computationally expensive than exact graph matching. Instead, we present a matching algorithm based on statistical hypothesis testing with low computational complexity. First we review the relevant concepts and notation then discuss our approach in these terms. An exhaustive introduction to hypothesis testing is outside the scope of this paper and so our discussion makes some simplifications.

A *hypothesis test* is a statistical method to determine whether a *null hypothesis* H_0 or *alternative hypothesis* H_a best explain some data [13]. The null hypothesis commonly theorizes the data is a result of chance and is accepted or rejected at a significance level α using some statistical test. If rejected, H_a is accepted as true with α probability of a type-I error (false positive).

A type of hypothesis test called a *goodness-of-fit* test can be used to determine the equality of probability distributions. We use the two-sample Kolmogorov-Smirnov (KS) test [14], first computing the *D-statistic* for two empirical cumulative distribution functions $\hat{S}_m(x)$ and $\hat{S}_n(x)$:

$$D_{m,n} = \max_x |\hat{S}_m(x) - \hat{S}_n(x)|$$

where m and n are the total event counts of their respective distributions. We then compute the probability that differences in the distributions are due to chance (the *p-value*) and reject H_0 if this value is less than our threshold α:

$$P(D_{m,n} \geq D_O | H_0) < \alpha$$

for the observed statistic D_O [13]. Though defined theoretically for continuous distributions, a modified KS test can be used with discrete distributions [15] or an unmodified test can simply provide conservative p-values.

We present two applications of the KS test for pattern classification in the following sections.

3.4 Network Motifs

One approach to characterizing communication topologies is to describe global communication patterns in terms of their localized subgraphs. Those subgraphs that occur more often than would be expected in randomized networks are called *motifs*. Network theorists have studied motifs in a wide range of fields including biology [16], ecology [17], chemistry [18], and neuroscience [19]. Motifs in such real world networks are often described as building blocks with intuitive functional interpretations, and these networks are commonly characterized by their distribution of n-node motifs. Such intuitive interpretations may also exist for HPC algorithms, though we leave this to future work and instead focus on their use as classifiers.

Motif discovery can be divided into three subtasks: 1) counting subgraphs, 2) grouping equivalent subgraphs, and 3) determining which subgraphs are over-represented relative to some random graph model. Each of these steps is computationally expensive and so the process is often restricted to 3, 4, or 5 node subgraphs.

We use the FANMOD tool [20, 21], which provides an order of magnitude speedup over its predecessors for steps 1 and 3. Input graphs may contain node and/or edge data referred to as *colors*. A maximum of 7 edge colors may be used and so the MPI calls stored on the edges of our ARGs are grouped by different criteria. Single color graphs assign all MPI calls to the same group, while 2-color graphs distinguish between broadcast and point-to-point calls. Lastly, 3-color graphs divide point-to-point calls into send and receive groups. The tool is run multiple times to find 1-3 color motifs of size 3 and 1-2 color motifs of size 4. Intuitively, larger motifs and more colors should help distinguish different patterns of communication.

Over-representation of a subgraph is determined by its z-score:

$$z = \frac{N_O - N_R}{\sigma}$$

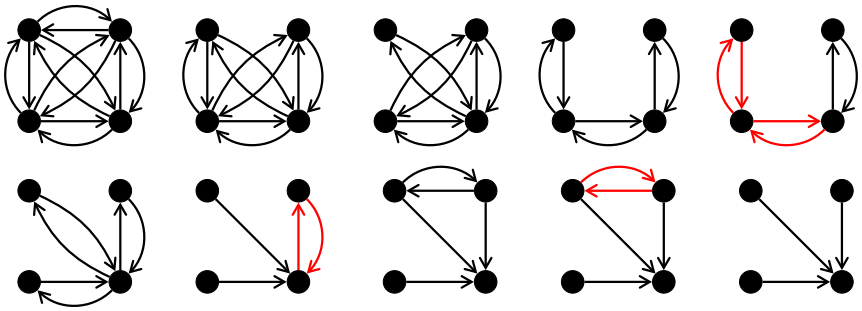


Figure 6: A random sample of common 4-node motifs with different edge colorings. Black represents all calls in single color motifs, while collective (black) and point-to-point (red) calls are represented separately in 2-color motifs.

where N_O is its count in the original network, N_R is its mean count in randomized networks, and σ is the standard deviation from N_R. For brevity, a random selection of the most common 3 and 4 node motifs are shown in Figures 5 and 6. These figures summarize topologically equivalent motifs that have the same number of colors but potentially different colorings. For example, the motif in the lower right of Figure 5 could represent two isomorphic motifs where red maps to a "send" call in the first and a "receive" call in the second.

We apply a KS test to compare the distribution over all motifs discovered in two communication graphs. If the p-value returned by the test is less than the significance level, the graphs and the parallel programs that generated them are considered distinct. Tests are performed for each pair of graphs to evaluate the accuracy of this approach and the results are presented in Section 3.6.

3.5 Call Distributions

Hypothesis testing is also used to test the fit of MPI call distributions between each corresponding rank of two parallel computations. The counts of MPI messages sent by each rank are normalized to form a probability distribution and the cumulative sum is taken to produce $\hat{S}$. For example: summed over all destination ranks, source rank 1 of program A may transmit 15 `MPI_Send`, 20 `MPI_Recv`, and 5 `MPI_Barrier` messages. The probability mass function is then 37.5%, 50%, and 12.5% respectively for these calls and 0% for all others. If source rank 1 of program B transmits 18 `MPI_Send`, 19 `MPI_Recv`, and 4 `MPI_Barrier` messages, the probability mass function is 43.9%, 46.3%, and 9.7%.

Taking their respective cumulative sums to obtain $\hat{S}_{40}$ ($m = 15{+}20{+}5$) and $\hat{S}_{41}$ ($n = 18{+}19{+}4$),

the KS test determines that these distributions are not significantly different at the $\alpha = 0.01$ level. If the probability mass functions remained the same but instead 2000 calls were logged, the test would determine the distributions are not the same at the $\alpha = 0.01$ level since there is far more data and the differences are less likely due to chance.

To determine if two communication patterns are generated by the same program, a KS test is applied to corresponding ranks in the communicators. For example: rank 1 of program A is compared to rank 1 of program B, rank 2 compared to rank 2, and so on. If the communicators are of the same size then all ranks are compared; if they are different, comparisons are performed between ranks present only in the smallest communicator. If more than some threshold of ranks are equivalent at significance level α, the programs are deemed equivalent. Both parameters provide an adjustable tolerance to topological differences. We found half the size of the smallest communicator to be an effective threshold.

3.6 Evaluation

A total of 328 logs (34 gigabytes) were collected for Lawrence Berkeley National Laboratory by the National Energy Research Scientific Computing Center (see Figure 7). Multiple logs exist for each program with varying ranks, parameters, architectures, and datasets when possible. Several simpler codes were logged by us; codes requiring significant domain knowledge or private datasets were logged from willing specialists on production systems. As a result, the inputs and parameters for some codes were not under our control. However, this dataset is several orders of magnitude larger than related efforts and we believe contains a representative sample of the dwarfs found in scientific computing.

Classifiers are evaluated by their true positive and false positive rates. A *true positive* (TP) denotes matching patterns generated by different runs of the same program; a *false negative* (FN) occurs when these patterns do not match. Similarly, a *true negative* (TN) occurs when patterns generated by different programs do not match; a *false positive* (FP) occurs when they do. The *true positive rate* (TPR) is defined as:

$$tpr = \frac{tp}{tp + fn}$$

Code	Area	Computational Dwarf	Communicator Size(s)
CACTUS	Astrophysics	Structured Grids	64, 256
FVCAM	Atmospheric Dynamics	Structured Grids	64
GTC	Magnetic Fusion	Unstructured Grids	64, 256
HYPERCLAW	Gas Dynamics	Structured Grids	256
LBMHD	Fluid Dynamics	Structured Grids	64, 256
MADBENCH	Benchmark	Dense Linear Algebra	256
MAESTRO	Hydrodynamics	Structured Grids	8, 512, 2048
MHD	Plasma Physics	Structured Grids	256, 512, 1024
MILC	Lattice Gauge Theory	Structured Grids	64, 256
NAMD	Molecular Dynamics	n-Body Methods	32, 64, 128
NPB	Benchmark	Multiple	64
PARATEC	Materials Science	Spectral Methods	64, 256
PDGEMM	Linear Algebra	Dense Linear Algebra	64
PDSYEV	Linear Algebra	Dense Linear Algebra	64
PF_2	Plasma Physics	Dense Linear Algebra	64, 256, 1024
PMEMD	Molecular Dynamics	n-Body Methods	64, 256
PSTG3R	Atomic Physics	Dense Linear Algebra	48, 196, 432, 768
SUPERLU	Linear Algebra	Sparse Linear Algebra	64, 256
SWEEP3D	Neutron Transport	Structured Grids	8

Figure 7: Summary of MPI codes used to generate IPM logs. Some codes may fall under multiple dwarf classes. Monte Carlo dwarfs are not represented as they are embarrassingly parallel and thus require minimal communication.

Similarly, true negatives (tn) and false positives (fp) define the *false positive rate* (FPR):

$$fpr = \frac{fp}{fp + tn}$$

The TPR and FPR for classifiers using subgraph isomorphism testing, motif distributions, and call distributions are shown in Figure 8.

Several results are of note. First, as mentioned, isomorphism testing is ineffective due to variance across multiple runs of the same program. However, hypothesis testing is effective both for 4-node motifs as well as call distributions. Given the increase in true positives moving from 3 to 4 nodes, a similar increase may be possible with 5 node motifs. Unfortunately, mining motifs of this size was computationally prohibitive.

Grouping related calls using finer-grained edge coloring appears to be beneficial. Additional colors could distinguish synchronous and asynchronous calls, or instead assign unique colors to each call. The latter case risks over-fitting the data: algorithms using slightly different calls for the

Classifier	TPR	FPR	Runtime
Subgraph Isomorphism	26.6	0.0	17m40s
Motif, 3 node, 1 color	56.9	8.3	Hours
Motif, 3 node, 2 color	52.5	4.2	Hours
Motif, 3 node, 3 color	51.3	3.9	Hours
Motif, 4 node, 1 color	77.6	2.2	Days
Motif, 4 node, 2 color	79.0	2.3	Days
Call Distribution	92.3	0.5	1m37s

Figure 8: Comparison of true positive rate (TPR) and false positive rate (FPR) for multiple communication pattern classifiers.

same computation would no longer have the same motif, and Figure 8 demonstrates that additional colors do not always improve accuracy. The number of colors required for such an evaluation is not currently supported by FANMOD.

Finally, while motifs are nearly as effective as call distributions, the time required for motif discovery is prohibitive for many applications, taking days or even weeks as opposed to minutes. As a result, some graphs (in particular those with 512 or more ranks) were excluded from the motif analysis.

Given its superior true and false positive rates and orders of magnitude less computation time (see Figure 8), hypothesis testing of MPI call distributions is our preferred method of classifying the communication patterns of distributed memory parallel programs. It is worth emphasizing the accuracy of the call distribution approach despite its time-independence; the overhead required to collect time-ordered data was prohibitive for our production systems.

It should be noted that these results occur in a more difficult multi-class classification setting: the chance of randomly guessing the correct class is only 7% here as opposed to 50% in a binary setting. Classification difficulty is also increased by the topological variance induced by changes in parameters, datasets, communicator size, and other factors. In particular, many programs exhibit all-to-all communication at some phase of execution, and this rules out the use of simpler classification methods based solely on topology with no notion of call names or other attributes.

4 Related Work

IPM logs have previously been used to study the performance of MPI applications. Fürlinger et al. [22] provide a general introduction to the IPM package and discuss several concepts related to this work including visualization of adjacency matrices and examining the distribution of aggregate MPI calls. Shalf et al. [23] perform similar analysis to evaluate the communication requirements of parallel programs for improving processor interconnect designs. The adjacency matrices of several NAS parallel benchmarks, augmented by number of messages and message size, are presented by Riesen [24].

Ma et al. [25] introduce a communication correlation coefficient to characterize the similarity of parallel programs using several metrics. The first compares the average transmission rate, message size, and unique neighbor count for each rank, while the second computes the maximum common subgraph. Their evaluation was limited to 4 programs in the NAS parallel benchmark.

In addition to graph and network theory, we have previously used machine learning to classify parallel computation [26, 27] and discuss related machine learning efforts elsewhere. This approach achieves greater accuracy than those found in Section 3.3 but requires substantially more computation and care to prevent overfit models.

Other parallel programming standards such as OpenMP are based on a shared, as opposed to distributed, memory model. In an effort to increase the portability of parallel software, recent work uses compiler techniques to translate OpenMP into MPI source code [28, 29], and our approach should apply when such techniques are used. While we focus on latent analysis using only runtime communications, source code translation has also been used to replace inefficient computations [30, 31]. The classification of communication patterns thus has strong ties to compilers, static analysis, and code optimization.

5 Conclusion

This work applies methods from graph and network theory to identify the latent class of a parallel computation from the observable information passed between nodes in a computational network: given logs of MPI messages from an unknown program, the task is to infer the program most likely to have generated those logs. Our original motivation was the detection of anomalous behavior on

HPC systems, though we present our work in a general context and suggest additional applications including performance analysis and automated algorithm replacement.

As initially postulated by work on computational dwarfs [3, 4], communication patterns tend to be highly structured and reflect the distributed memory access patterns of the underlying algorithm. When dealing with algorithm *implementations*, however, many other factors affect the communication patterns of theoretical algorithms. Different implementations of the same algorithm, shared libraries, compiler optimizations, architecture differences, software flaws, debug flags, and numerous MPI implementations all make this task more difficult. Further, some parallel programs have data-dependent communication topologies, varying both slightly (see Figure 3) and greatly as with multi-use ("swiss-army") libraries or interpreters such as Matlab.

Using gigabytes of data covering over a dozen parallel programs, we constructed directed communication graphs and found network-theoretic measures such as node degree and centrality distributions capture insufficient information on their own to classify parallel computations. We also examined subgraph isomorphism testing for comparing topologies with different numbers of compute nodes and found many topologies exhibit the worst-case factorial runtime of the algorithm. More importantly, isomorphism requires exact matching and thus lacks the error tolerance necessary to correctly classify data-dependent computation.

To perform approximate matching we evaluate two approaches using goodness-of-fit tests; the first using the distribution of over-represented subgraphs called motifs and the second using the distribution of MPI calls relative to each rank. Both approaches succeed in classifying data-dependent topologies and comparisons are extremely fast excluding the motif discovery phase. A significance level allows tuning false positive and false negative rates. The best method, comparing per-rank distributions of MPI calls, achieved a 92% true positive rate in less than 2 minutes. However, swiss-army programs with extreme pattern variance elude this approach.

Other statistical measures such as Claussen's offdiagonal complexity [32] may also be useful for approximate topology comparison. Graph edit distance [5], Bayesian [33] and spectral [34] approaches to edit distance, graph kernels [35], and factor graphs [36] offer additional approaches to approximate graph matching.

These directions may provide increased generality for topologies not yet observed or performance bounds in adversarial environments [37]. However, the results presented in this paper show that

error-tolerant methods for matching parallel communication patterns are practical for inferring latent classes of computation.

6 Acknowledgements

Thanks to Scott Campbell and David Skinner for capturing IPM data at NERSC and to members of the high performance computing security project at LBNL for helpful discussions. This research was supported in part by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC02-05CH11231, and also by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001 under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College. The views and conclusions contained in this document are those of the authors and not necessarily those of its sponsors.

References

[1] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, S. Kumar, E. Lusk, R. Thakur, and J. L. Träff, "MPI on a Million Processors," in *Proceedings of the 16th European PVM/MPI Users' Group Meeting*, pp. 20–30, 2009.

[2] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas, "Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms," in *Proceedings of the 2005 International Conference on Parallel Processing*, pp. 119–128, 2005.

[3] K. Asanović, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, "The Landscape of Parallel Computing Research: A View From Berkeley," Tech. Rep. UCB/EECS-2006-183, University of California, Berkeley, 2006.

[4] P. Colella, "Defining Software Requirements for Scientific Computing," tech. rep., DARPA High Productivity Computing Systems, 2004.

[5] A. Sanfeliu and K. Fu, "A Distance Measure Between Attributed Relational Graphs for Pattern Recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, no. 3, pp. 353–362, 1983.

[6] L. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[7] D. B. West, *Introduction to Graph Theory*. Prentice Hall, 2nd ed., 2001.

[8] S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pp. 151–158, 1971.

[9] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[10] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[11] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), pp. 11–16, 2008.

[12] P. Foggia, "The VFLib Graph Matching Library, version 2.0," 2001.

[13] J. D. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference*. CRC Press, 5h ed., 2010.

[14] F. J. Massey, "The Kolmogorov-Smirnov Test for Goodness of Fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[15] W. J. Conover, "A Kolmogorov Goodness-of-Fit Test for Discontinuous Distributions," *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 591–596, 1972.

[16] U. Alon, "Network Motifs: Theory and Experimental Approaches," *Nature Reviews Genetics*, vol. 8, no. 6, pp. 450–461, 2007.

[17] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network Motifs: Simple Building Blocks of Complex Networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[18] M. Kuramochi and G. Karypis, "Frequent Subgraph Discovery," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, pp. 313–320, 2001.

[19] J. Qian, A. Hintze, and C. Adami, "Colored Motifs Reveal Computational Building Blocks in the C. elegans Brain," *PLoS ONE*, vol. 6, no. 3, p. e17013, 2011.

[20] S. Wernicke and F. Rasche, "FANMOD: A Tool for Fast Network Motif Detection," *Bioinformatics*, vol. 22, no. 9, pp. 1152–1153, 2006.

[21] S. Wernicke, "Efficient Detection of Network Motifs," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 347–359, 2006.

[22] K. Fürlinger, N. J. Wright, and D. Skinner, "Effective Performance Measurement at Petascale Using IPM," in *Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems*, pp. 373–380, 2010.

[23] J. Shalf, S. Kamil, L. Oliker, and D. Skinner, "Analyzing Ultra-Scale Application Communication Requirements for a Reconfigurable Hybrid Interconnect," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[24] R. Riesen, "Communication Patterns," in *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, pp. 275–282, 2006.

[25] C. Ma, Y. M. Teo, V. March, N. Xiong, I. R. Pop, Y. X. He, and S. See, "An Approach for Matching Communication Patterns in Parallel Applications," in *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–12, 2009.

[26] S. Peisert, "Fingerprinting Communication and Computation on HPC Machines." 2010.

[27] S. Whalen, *Security Applications of the e-Machine*. PhD thesis, University of California, Davis, 2010.

[28] A. Basumallik and R. Eigenmann, "Towards Automatic Translation of OpenMP to MPI," in *Proceedings of the 19th International Conference on Supercomputing*, pp. 189–198, 2005.

[29] A. Basumallik, S. Min, and R. Eigenmann, "Programming Distributed Memory Sytems Using OpenMP," in *Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 207–214, 2007.

[30] R. Metzger and Z. Wen, *Automatic Algorithm Recognition and Replacement: A New Approach to Program Optimization*. MIT Press, 2000.

[31] R. Preissl, M. Schulz, D. Kranzlmüller, B. R. de Supinski, and D. J. Quinlan, "Transforming MPI Source Code Based On Communication Patterns," *Future Generation Computer Systems*, vol. 26, no. 1, pp. 147–154, 2010.

[32] J. C. Claussen, "Offdiagonal Complexity: A Computationally Quick Complexity Measure for Graphs and Networks," *Physica A*, vol. 375, no. 1, pp. 365–373, 2007.

[33] R. Myers, R. C. Wison, and E. R. Hancock, "Bayesian Graph Edit Distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 628–635, 2000.

[34] A. Robles-Kelly and E. R. Hancock, "Graph Edit Distance from Spectral Seriation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 3, pp. 365–378, 2005.

[35] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman Graph Kernels." 2010.

[36] J. Reichardt, R. Alamino, and D. Saad, "The Interplay between Microscopic and Mesoscopic Structures in Complex Networks," *PLoS ONE*, vol. 6, no. 8, p. e21282, 2011.

[37] M. Barreno, P. L. Bartlett, F. J. Chi, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, U. Saini, and J. D. Tygar, "Open Problems in the Security of Learning," in *Proceedings of the 1st ACM Workshop on AISec*, pp. 19–26, 2008.