High Performance and Energy Efficient Wireless NoC-Enabled Multicore Architectures for Graph Analytics

Karthi Duraisamy, Hao Lu, Partha Pratim Pande, Ananth Kalyanaraman

School of EECS
Washington State University
Pullman, WA 99164, U.S.A.
{kduraisa, hlu, pande, ananth}@eecs.wsu.edu

Abstract

With its applicability spanning numerous data-driven fields, the implementation of graph analytics on multicore platforms is gaining momentum. The most important component of a multicore chip is its communication backbone. Due to the inherent irregularities in data movements manifested by graph based applications, it is essential to design an efficient on-chip interconnect for multicore chips performing graph analytics. In this paper we present a detailed analysis of the traffic patterns generated by graph-based applications when mapped to multicore chips. Based on this analysis, we present the design of wireless Network-on-Chip (WiNoC)-enabled multicore platforms for efficient implementation of graph analytics. When compared to traditional wireline mesh architecture, WiNoC enables a faster data exchange among the computing cores, leading to reduced execution times and lower energy dissipation. We demonstrate that depending on the particular graph application, the WiNoC reduces the execution time up to 35% and lowers the energy dissipation up to 40% when compared to traditional wireline mesh.

Categories and Subject Descriptors

C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Interconnection architectures, G.2.2 [**Graph Theory**]: Graph algorithms, network problems, B.4.4 [**Performance Analysis and Design Aids]**: Simulation, C.2.1 [**Network Architecture and Design**]: Wireless communication.

General Terms

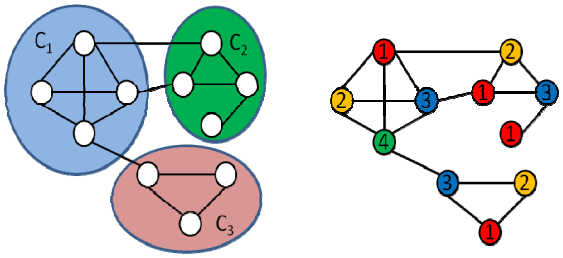
Algorithms, Performance, Design, Experimentation.

Keywords

Network-on-Chip architectures; Graph Analytics; Wireless NoCs; Community detection; Graph coloring

1. Introduction

The prevalence of multicore architectures opens new opportunities of running data-parallel applications on a single chip instead of using large clusters. In an era when power constraints and data movement are proving to be significant barriers for the application of high-end computing, multicore architectures offer a low-power and high bandwidth platform suitable for both data- and compute-intensive applications. The Network-on-Chip (NoC) paradigm has emerged as a revolutionary methodology for integrating a very high number of embedded cores in a single die. The existing method of implementing a NoC with planar metal interconnects is deficient due to high latency and significant power consumption arising out of multi-hop links used in data exchanges. In this context, prior work has already established the high potential of using wireless NoC (WiNoC) for energy-efficient multicore platforms [1]. In a related work [2], WiNoC was used to build an on-chip platform with *small world* connectivity characteristics. This platform was demonstrated to be highly efficient in handling data transfers on a multicore system.



(a) Community Detection **(b) Graph Coloring**

Figure 1. An illustrative example for the two different graph operations addressed in this paper – viz. community detection and graph coloring. Given a graph G(V,E), part (a) shows a possible community-wise partitioning of vertices in V. Part (b) shows a 1-distance coloring of vertices, with the numbers within each vertex identifying the color assignment.

There is some effort in designing multicore chips customized for emerging big-data workloads [3]. However, many data-driven applications use complex graph representations and yet there has not been any effort to study NoC-based architectures for graph analytics.

In this work, we demonstrate that a WiNoC-enabled multicore chip can achieve significant improvement in execution time and energy dissipation for solving large-scale graph applications. Achieving parallel scalability in graph applications remains a significant challenge due to the inherent irregularity in real world networks that in turn causes irregularities in computation and data movement. Consequently, mapping graph-theoretic applications on modern day multicore architectures with support for fast latency-reducing and energy-minimizing on-chip network interconnects will be key to executing large-scale graph operations at scale.

In this paper, we focus on two graph operations – community detection and graph coloring – as two exemplar operations in

advanced graph analytics. Figure 1 shows an illustrative example for each of these two graph operations.

Let $G(V,E,\omega)$ be an input graph, where V is the set of vertices, E is the set of edges, and ω is a function that maps every edge to a numeric weight. Given $G(V,E,\omega)$, the *community detection* problem [4] is one of partitioning the set of vertices in V into "communities" such that the modularity of the partitioning [5] is maximized. Modularity is a measure between 0 and 1 and it reflects the ratio between the net weights of intra-community edges to inter-community edges. Neither the number of communities nor the size distribution of communities is known *a priori*. In fact, community detection is used to reveal such natural divisions that exist in real world networks. It is used in a number of scientific applications including (but not limited to) social network analysis, bioinformatics, collaboration networks, and electric power grid [4]. Despite its broad application base, executing community detection over large real world graphs remains a challenging problem despite recent developments in multicore processing. The heuristic nature of algorithms alongside the need to repeatedly access neighbourhoods of vertices in an irregular fashion causes irregular data access and movement patterns that impede performance and scalability in traditional multicore environments.

The second graph operation that we consider is graph coloring. The goal of *k-distance graph coloring* is as follows: Given an input graph G(V,E), assign colors to vertices such no two vertices separated by k hops are assigned the same color [6]. In this paper, we focus on 1-distance coloring, where any two adjacent vertices in the graph are to be assigned different colors. Coloring is a classical graph operation that is widely used in a number of graph-based scientific applications to determine compatible parallel schedules [7], [8]. As edges in graphs represent node-to-node interdependencies, coloring can be used to obtain a parallel schedule that guarantees no two vertices that are interdependent on one another are processed during the same parallel step (i.e., same "color"). However, such an approach needs to also have as few parallel steps (or color classes) as possible, and therefore a second goal for graph coloring is to minimize the number of colors used in assignment. In addition, since concurrency is limited by the number of vertices within each color class, there is also a need to ensure load balanced color distribution and this variant has been extensively studied under the context of equitable and balanced coloring [9], [10]. Recently, coloring implementations have been proposed to obtain an initial coloring so as to minimize the number of colors, and subsequently redistribute the vertices among color classes so as to obtain a balanced coloring [11]. The assignment of colors to vertices and their redistributions make the graph coloring operation data movement-bound and lock-intensive in traditional multicore environments. In addition these characteristics heavily depend on the underlying graph structure and connectivity.

The irregular data access patterns and memory-bound nature of community detection operation, and the communication-bound and lock-intensive nature of the graph coloring operation represent two different and challenging use-cases for multicore parallelism. More specifically, in this paper we explore Wireless Network-on-Chip (WiNoC)-enabled multicore architectures to efficiently implement these two graph operations. With its efficient on-chip interconnect (wireline for short range traffic and wireless of long range traffic), and the capacity to integrate a huge number of cores on a single chip, WiNoC-enabled multicore architecture presents an attractive platform for optimally implementing irregular data-bound graph-based operations. More specifically, the contributions of this paper include:

1. Analyses of the on-chip traffic patterns generated by community detection and balanced graph coloring applications on real world graphs;
2. Design and implementation of WiNoC-based multicore platforms fine-tuned for efficient execution of community detection and balanced graph coloring;
3. Experimental evaluation of small world network-based WiNoC and comparative performance evaluation with respect to the conventional wireline mesh topology.

To the best of our knowledge, our work represents the first detailed study of advanced graph operations on WiNoC-enabled multicore systems.

2. Related Work

Designing specialized computation architectures and parallel algorithms for big data analytics has been an area of great interest in recent times. A detailed study on using microarchitectures for graph analytics has been presented in [12]. In [13], the performance of Symmetric Multiprocessor (SMP) systems for large-scale graph analytics has been analyzed. For the two target graph operations of community detection and graph coloring, multithreaded implementations for traditional multicore (x86) architectures and also many-core architectures such as Tilera chips have been previously [11], [14], [15], [16], [17], [18] developed.

In [19], a MapReduce based parallel computing model for large graph analytics has been presented. This model has been shown to be particularly efficient for two popular graph operations, 3-clique enumeration of a graph and computation of clustering coefficient. Possibilities of reducing on-chip interconnect energy for MapReduce applications through the use of dynamic directories have been presented in [20]. The dynamic directories reduce the exchange of network control messages, leading to energy conservation.

As stated in section 1, NoCs have emerged as standard communication backbones for multicore chips. Conventional NoCs use multi-hop, packet switched communication. Design and optimization of multi and many-core systems on chip that exploit small world effects have been already demonstrated in [21]. A comprehensive survey regarding various WiNoC architectures is presented in [1], which shows the possibility of creating novel architectures enabled by on-chip wireless links.

3. Introduction to Coloring and Community Detection

3.1 Grappolo

Recently, Lu et al. [14] developed a scalable parallel implementation for executing community detection on conventional multicore architectures. This method, called *Grappolo*, implements a multi-phase, multi-iterative heuristic to maximize the modularity of the output partitioning. In what follows, we describe the main algorithmic details and data structures of Grappolo.

Given an input graph $G(V,E,\omega)$, containing n vertices and m edges, the algorithm executes multiple *phases*, and multiple *iterations* within each phase, until convergence. Within each phase the following steps are carried out:

1) Initially, each vertex is assigned to an individual community of its own.
2) Within each iteration the vertices are scanned in parallel and for each vertex, a decision is made to determine whether or not to migrate it to one of its neighbouring communities (as defined by the communities of its neighbours). To avoid

locking, the parallel implementation uses community assignments as of the previous iteration. Other graph heuristics are used to resolve potential conflicts and in effect to ensure that the net modularity gain achieved by the end of the iteration always remains non-negative.

3) The algorithm proceeds this way from one iteration to the next until the net modularity gain is no longer positive or is smaller than a certain specified cut-off. When this convergence criterion is met, the algorithm terminates the current phase. At this stage, a *compacted graph* $G'(V',E',\omega')$ is computed by compacting every community detected in G to a single meta-vertex in G', with the new edges and their weights reflecting the weights of intra- and inter-community edges of G.
4) Subsequently, the algorithm initiates the next phase on the newly compacted graph G', until no more appreciable modularity gain is achieved.

The main data structures used in Grappolo are as follows: The graph is stored as two arrays of size $O(m+n)$ in the compressed sparse row (CSR) format [22]. Each vertex entry also stores the current community assignment for that vertex. In addition a separate array data structure is used to keep track of the degree of each community (i.e., sum of the degree of all vertices belonging to each community). The orders in which the vertices and communities are stored in their respective arrays are arbitrary as it is not possible to predetermine locality properties due to the dynamic nature of the algorithm.

3.2 Balanced coloring

Given an input graph $G(V,E)$, the goal of 1-distance coloring is to determine a color assignment to vertices such that adjacent vertices are always assigned different colors. A classic greedy coloring heuristic used for minimizing the number of colors used can be described as follows [16]: For each vertex $v \in V$ (in some order), assign a color that is not yet used by any of its neighbouring vertices. Typically the first available color index is used for this purpose (assuming the colors are identified numerically), and if all colors are used by the neighbours, a new color is introduced. This heuristic represents a Greedy First Fit approach and has been shown in practice to be highly effective in reducing the number of colors used. However, it also typically creates a highly skewed color size distribution.

In a recent work, Lu et al. [11] proposed a number of approaches to generate a balanced coloring – i.e., the number of vertices assigned per color class (aka. color bin) is roughly the same. They implemented a parallel approach whereby the Greedy First Fit approach is used to first obtain an *initial coloring* of the graph, and a subsequent balancing step redistributes the vertices among the different bins so as to obtain a balanced coloring without increasing the overall number of colors used. Different balancing schemes were explored and one of the more effective schemes, namely *Greedy-CLU*, works as follows. Color classes are processed one at a time, such that the vertices within every over-full color bin (i.e., those containing more than the mean number of vertices) are redistributed to one or more under-full bins until the number of vertices within the source bin reaches the target mean or no more valid vertex moves are possible. The choice of the under-full bin is performed greedily by selecting a smallest available bin that is also compatible with the vertex being moved (i.e., contains no neighbours of that vertex). For this reason, this scheme is labelled a greedy color bin-based Least Used approach (*Greedy-CLU*).

This implementation also uses the CSR format for storing the input graph, while a separate array-based data structure is used to keep track of the color bins and their sizes. Experimental evaluation showed this scheme to be effective in generating highly balanced color bins. However, the parallel scalability of the overall coloring process is affected by the need to lock the coloring data structures in order to update the source and destination bin sizes. In addition, the need to perform compatibility checks for vertices requires to access color bins corresponding to the neighbours of every vertex. This step introduces irregular memory access and workloads.

Thus the balanced coloring application involves two distinct phases, an *initial coloring phase* and a *redistribution phase*, between which the redistribution phase is a communication-bound phase with heavy vertex migration traffic and locking. Furthermore, the distribution phase is devoid of any complex computations. Hence, unlike the initial coloring phase, the execution speed of the redistribution phase is highly dependent on the speed of the data migration.

Henceforth, for ease of exposition we will refer to the above two specific multicore algorithms for community detection and balanced coloring as simply "Grappolo" and "Balanced Coloring", respectively. These two algorithms will serve as our target implementations for the WiNoC-based multicore platform.

4. Wireless NoC Architecture

Both community detection using Grappolo and balanced coloring applications exhibit inherent irregularities due to data movements. As stated in section 3, the computation within Grappolo has multiple phases with alternating parallelized (clustering) and serialized (compaction) characteristics. In addition, Grappolo involves migration of vertices between the communities during each clustering phase. In contrast, balanced coloring consists of two major parallel phases, initial coloring and redistribution, with bulk of data movement and locking related traffic occurring during the redistribution phase.

Due to these application characteristics, the two graph applications generate substantial long-range traffic patterns while executing on a multicore platform (as later shown in Figure 3), with most of these data exchanges happening between computing cores that are physically far apart. Furthermore, these graph applications show the presence of one or more hotspot cores whose traffic injection rates are much higher than that of the average traffic injection rate (as later shown in Figure 4). Due to these long-range and skewed traffic patterns, we posit that designing an

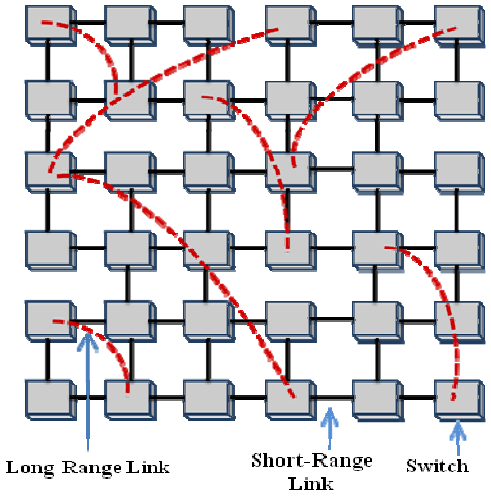


Figure 2. An illustration of the WiNoC Architecture for Graph analytics

on-chip interconnect infrastructure that enables low-latency data exchange, even among physically distant cores will be critical to achieving performance at scale in these graph applications. Furthermore, the on-chip interconnect should be able to efficiently redistribute the traffic from the hotspot regions.

Considering all these facts, we present the design of a small world network-based WiNoC architecture for graph analytics. In this WiNoC architecture each computing core is connected to a switch and switches are interconnected using wireline and wireless links, as shown in Figure 2. In the WiNoC, the overall interconnection network is designed using a power-law model [23]. In this power-law based network, there are several long-range communication links. When implemented with conventional metal wires, these links are extremely costly in terms of power and delay. Hence, in WiNoC, more power efficient wireless links are used to interconnect the switches that are separated by a long distance. We use the mm-wave wireless links operating in the 10-100 GHz range here. In [1], it is demonstrated that it is possible to create three non-overlapping wireless channels with on-chip mm-wave wireless transceivers. Using these three channels we overlay the wireline small-world connectivity with the wireless links such that a few switches get an additional wireless port. The wireless ports have a Wireless Interface (WI) tuned to one of the three different frequency channels. For a 64-core system, WiNoC uses twelve WIs in total with four WIs operating on each of the 3 wireless channels [2].

4.1 Placement of Wireless Nodes

In the WiNoC creation process, the wireless node placement strategy focuses on minimizing the optimization metric μ, which is defined as:

$$\mu = \Sigma_{\forall i}\Sigma_{\forall j} f_{ij} h_{ij} \quad (1)$$

Here h_{ij} denotes the minimum distance in number of hops from switch i to switch j with the given network connection. The f_{ij} value denotes the frequency of interaction between the switches. The f_{ij} values are determined by analyzing the traffic patterns generated while executing the Grappolo and Balanced Coloring applications with a set of test graphs.

To optimize the network for handling multiple applications, a combined f_{ij} metric can be used. Let L denote the number of application instances used during our training runs. Then, the interaction frequency is given by:

$$f_{ij} = \frac{1}{L}\sum_{n} \frac{f_{ijk}}{\sum_{\forall i}\sum_{\forall j} f_{ijk}}, \quad k = 1,2 \ldots L \quad (2)$$

where f_{ijk} denotes the frequency of interaction between switches i and j for a specific application instance k.

4.2 Routing Protocol

The power-law model based WiNoC principally has an irregular network topology. Irregular networks require topology agnostic routing methods. In this work, we use ALSAH protocol [24] for routing packets in WiNoC. ALASH is built upon the layered shortest path (LASH) algorithm, but has more flexibility by allowing each message to adaptively switch paths, letting the message choose its own route at every intermediate switch. The LASH algorithm takes advantage of the multiple virtual channels in each switch port of the NoC switches in order to route messages along the shortest physical paths. In order to achieve a deadlock-free operation, the network is divided into a set of virtual layers, which are created by dedicating the virtual channels from each switch port into these layers. The shortest physical path between each source-destination pair is then assigned to a layer such that the layer's channel dependency graph remains free from cycles. A message is not allowed to revisit a layer that it has previously traveled in order to prevent deadlocks.

4.3 Wireless MAC protocol

The WiNoC uses a distributed MAC protocol to resolve the channel access contention among the wireless nodes [25]. The wireless channel used in the WiNoC is a shared medium. Hence, each wireless node inherently knows if there is any on-going wireless transmission. In the WiNoC, if a wireless node has messages to transmit and if the wireless channel is free, the wireless node first broadcasts a request to acquire the wireless channel. The request packets follow a simple orthogonal on-off keying mechanism. This enables multiple wireless nodes to simultaneously transmit the request packets. Once all the broadcasted request packets are received, they are simultaneously processed by all the wireless nodes using a common node selection algorithm. After request processing, one of the requested wireless nodes acquires the channel and proceeds with data transmission.

4.4 Wireless Interface

The two principal wireless interface components for the WiNoC architecture are the antenna and the transceiver. WiNoC uses a metal zigzag antenna that has been demonstrated in [2], as it provides the best power gain with the smallest area overhead. A detailed description of the transceiver circuit is out of the scope of this paper. However, the transceiver was designed and fabricated and all the details are provided in [2]. The wireless interface is completely CMOS compatible and no new technology is needed for its implementation. The wireless interface has an area overhead of 0.25 mm^2 per transceiver. For data rates of 16 Gbps, the wireless link dissipates 1.95 pJ/bit.

5. Experimental Results and Evaluation

5.1 Experimental Setup

Test platforms**:** In this section we evaluate the performance of a 64 core multiprocessor system running the Grappolo and Coloring applications. We consider the WiNoC and traditional wireline Mesh NoC architectures in this performance evaluation. The die size of the system under consideration is 20×20 mm^2. We use GEM5 [26], a full system simulator, to obtain detailed processor and network-level information. We consider a system of x86 cores running Linux within the GEM5 platform for all experiments. The memory system is MOESI_CMP_directory setup with private 64KB L1 instruction and data caches and a shared 16MB (256KB distributed per core) L2 cache. The width of all wireline links is considered to be the same as the flit width, which is considered to be 32 bits in this paper. Both wireline and wireless links follow wormhole routing. All switch ports except those associated with the Wireless Interfaces (WIs) have a buffer depth of two flits. The ports associated with the WIs have an increased buffer depth of eight flits to avoid excessive latency penalties while waiting for wireless channel access [2]. Energy dissipation of the network switches, inclusive of the routing and MAC blocks, was obtained from the synthesized netlist by running Synopsys™ Prime Power, while the energy dissipated by wireline links was obtained through HSPICE simulations taking into consideration the length and layout of wired links. The processor-level statistics

generated by the GEM5 simulations are incorporated into McPAT (Multicore Power, Area, and Timing) to determine the processor-level power values [27].

Test inputs: The evaluation of Grappolo is performed with the help of DIMACS10 clustering instance graph data sets [28]. We have considered five DIMACS10 clustering instance graphs, Hep-th (HEP), Astro-ph (ASTRO), Comd-mat-2003 (COND), PGPgiantcompo (PGP) and as-22july06 (ASJ), to evaluate the benefits achieved by the WiNoC interconnect in execution of Grappolo applications. The evaluation of the balanced coloring application is carried out with the help of six DIMACS10 graphs. Two street network graphs, Belgium (BLG) and Netherlands (NTH), three citation network graphs, citationCiteseer (CCR), coAuthorsDBLP (CAD), coAuthorsCiteseer (CAC) and one clustering instance graph cnr-2000 (CNR) are used for these evaluations. Due to the shorter runtimes for the balanced coloring application, we were able to test much larger data sets. Further information on the datasets analyzed is provided in Table 1 and 2.

Table 1. DIMACS10 [28] graphs used in Grappolo analyses.

Graph	Number of Vertices	Number of Edges	Number of Communities
ASJ	22,963	48,436	31
PGP	10,680	24,316	99
ASTRO	16,706	121,251	412
HEP	8,361	15,751	636
COND	31,163	120,029	945

Table 2. DIMACS10 [28] graphs used for Balanced Coloring analyses.

Graph	Number of Vertices	Number of Edges	Number of Colors
NTH	2,216,688	244,1238	5
BLG	1,441,295	154,9970	5
CCR	268,495	1156,647	17
CNR	325,557	2738,969	86
CAC	227,320	814,134	87
CAD	299,067	977,676	115

5.2 Analysis of Traffic Patterns

In this section we analyze the nature of the traffic patterns generated by Grappolo and Balanced Coloring. Figure 3 shows the percentage of total traffic exchanged between the communicating cores separated by certain distances for Grappolo and Balanced Coloring. From this figure we can observe that both Grappolo and Balanced Coloring generate significant amount of long-range traffic, although owing to different reasons. In case of Grappolo, long-range traffic is largely due to shared memory access during the clustering computation; while in the case of Balanced Coloring, it is due to locks generated during the vertex redistribution phase However, it is evident that irrespective of the graphs considered in both applications, 70% of the total injected messages are transferred among cores that are more than 7.5mm apart. It has been already shown that wireless links are more energy efficient than traditional metal wires when the link length exceeds 7.5mm in the 65 nm technology node [2].

Figure 4 shows the percentage of the traffic injected by each of the top six traffic hotspots present in the system, while executing Grappolo and Balanced Coloring. It can be observed from this figure that up to 10% of the total traffic is associated with a single core. Moreover, the top 6 traffic hotspots in Grappolo are responsible for about 25% of the total injected traffic. In Balanced Coloring, the top 6 traffic hotspots are responsible for up to 27% of

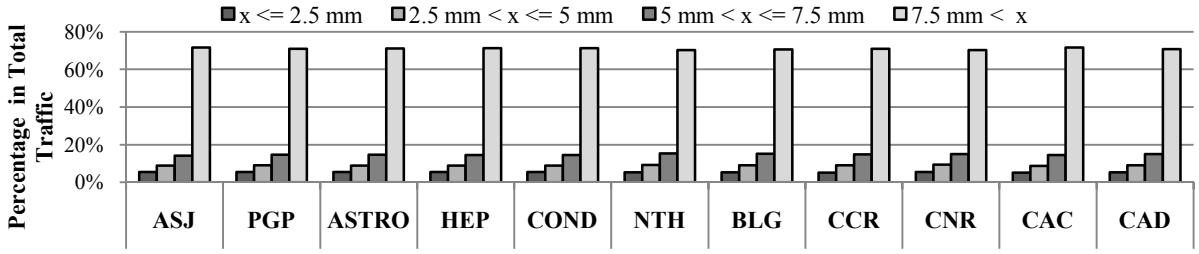


Figure 3. Fraction of total traffic exchanged between the communicating cores separated by distance x for Grappolo and Balanced Coloring. The values for x are specified in the legend.

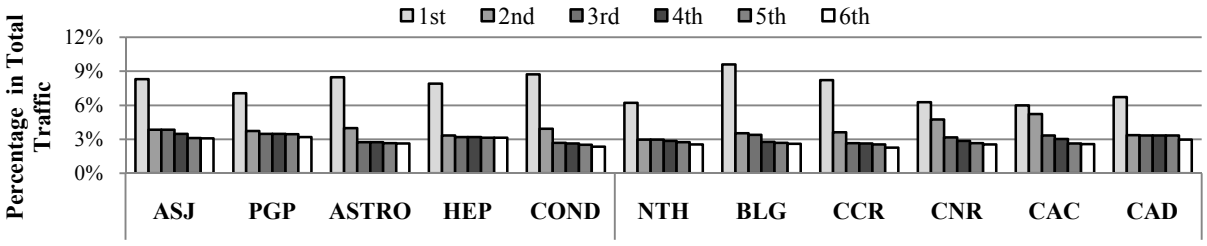


Figure 4. Traffic hotspot nodes: Percentage in total traffic injected by the top 6 traffic hotspots for Grappolo and Balanced Coloring.

the total system traffic. In case of Grappolo, this injection originates mostly from the master core, which is responsible for the graph compaction phase. Recall that the compaction step is largely serialized owing to the need to gather vertices belonging to communities.

In case of Balanced Coloring, the hotspots appear during the redistribution phase. This can be attributed to the overheads associated with the management of locks. During the redistribution phase, the processed vertices are widely distributed among the processing nodes. However, when multiple vertices located at different cores compete to acquire locks, the lock management introduces hotspots during acquisition and release of those locks.

The above results indicate that both Grappolo and Balanced Coloring are characterized by significant long-range communication patterns and traffic hotspots. Thus both of these present an ideal case for benefiting from the WiNoC architecture. Hence, in the next sections, we study the performance and energy dissipation profile of WiNoC-enabled multicore architectures running these graph analytics.

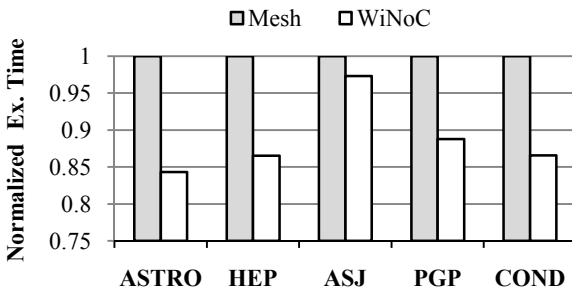
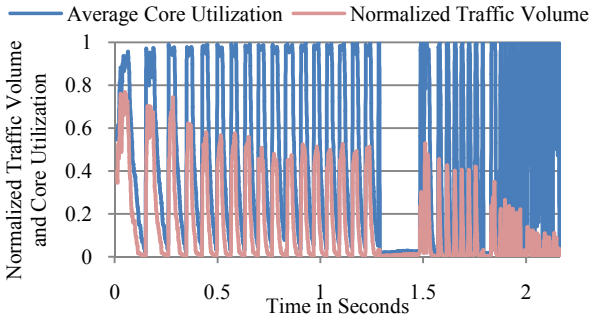
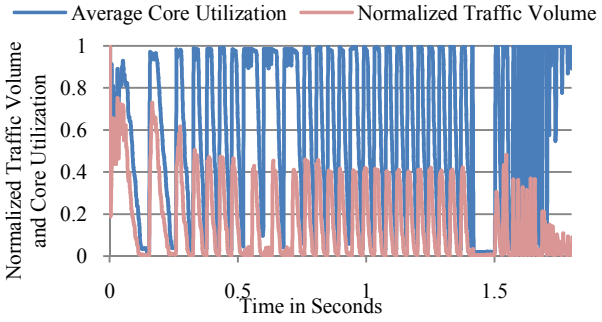


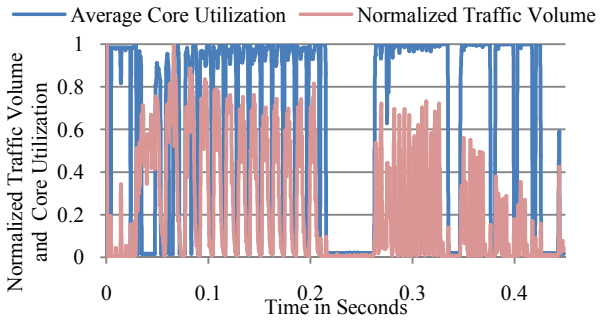
Figure 5. Grappolo Execution Times



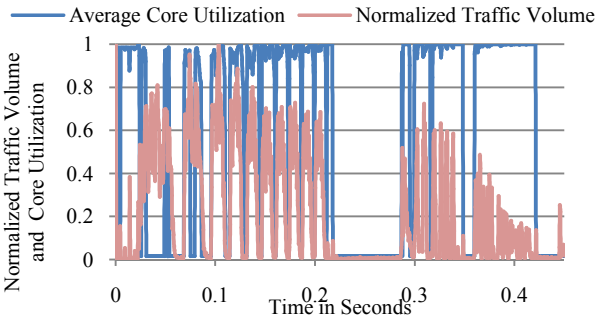
6(a). COND dynamic characteristics



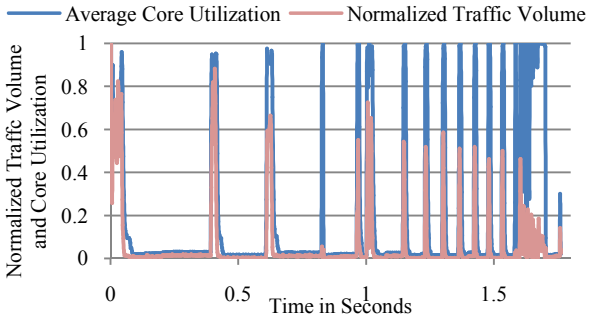
6(b). ASTRO dynamic characteristics



6(c). HEP dynamic characteristics



6(d). PGP dynamic characteristics



6(e). ASJ dynamic characteristics

Figure 6. Grappolo: Variation in Traffic Volume and average CPU activity over multiple clustering and compaction phases with WiNoC interconnection architecture

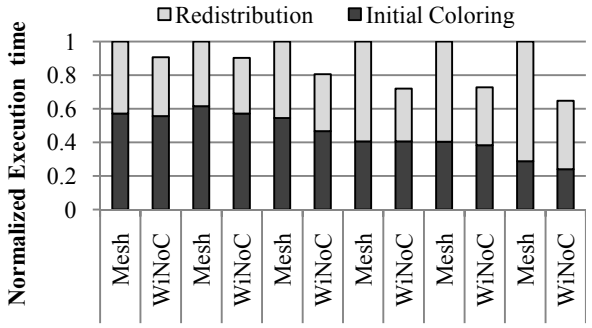


Figure 7. Balanced Coloring: Execution Times

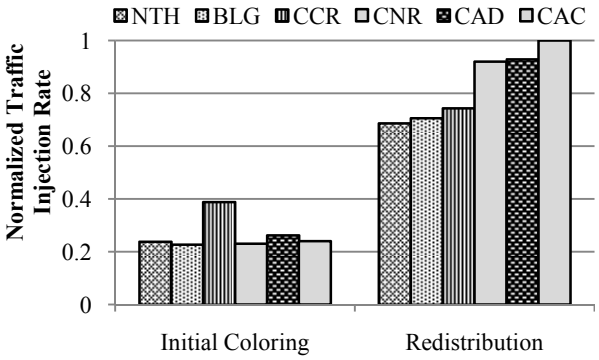


Figure 8. Balanced Coloring: Trafic Injection Rates

5.3 Execution Time Improvement

In this section we evaluate the runtime of our WiNoC architecture when executing the two target graph applications. We use the traditional wireline mesh network as the baseline for comparison.

5.3.1 Grappolo

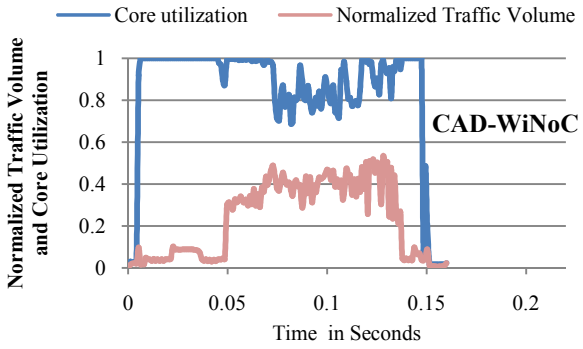
Figure 5 shows Grappolo's execution times on both WiNoC and mesh interconnected multicore systems. Among the five graphs considered, ASTRO achieves the highest execution time improvement (15.6%) by using the WiNoC. ASTRO is followed by HEP with 14.45% improvement and COND with 14.4% improvement. The ASJ graph achieves the least execution time improvement (3%). The reason for the variations in execution time improvement can be understood by analyzing the runtime characteristics of Grappolo. Figure 6 shows the varying traffic volumes and the average CPU utilizations, over the entire Grappolo execution period with the WiNoC architecture. Here, traffic volumes indicate the volume of data exchanged through the interconnection network. As discussed earlier, Grappolo involves multiple executions of two distinct operations, clustering and compaction. Since all computing cores are highly active during the clustering phase, it is identified with high average core utilizations. However, the master core is the only core that is active during compaction. This serialization within the compaction phase is identified with a lower average CPU utilization.

Since all the cores are active, inter-core traffic volumes associated with the clustering phase are much higher than that of the compaction phase. It can also be noted from Figure 6, that the traffic volume decreases with time during the clustering phase. As the execution of the community detection algorithm progresses, the number of clusters reduces, and consequently the interactions between the cores also begin to reduce, leading to the observed drop in traffic volume. Similar trends were observed with the mesh architecture albeit with longer execution times (as was observed in Figure 5).

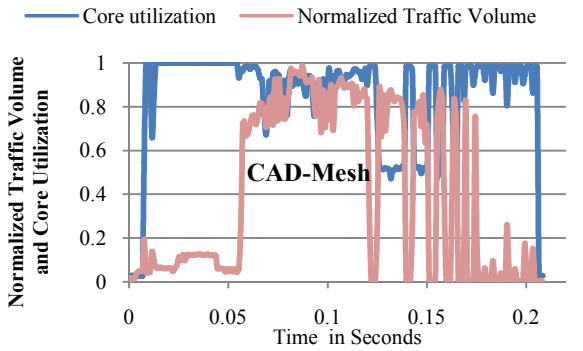
These variations in CPU utilization and network traffic among the above mentioned input graphs lead to the observed variation in runtime improvements achieved with WiNoC. By comparing Figures 5 and 6, we can note that the graphs with longer clustering durations achieve better execution time improvements through the use of WiNoC. Owing to its power-law based connectivity, the WiNoC has a much lower average hop count among the cores when compared to the traditional wireline mesh architecture [2]. Hence, when compared to mesh, the WiNoC enables a faster data exchange among the computing cores, leading to a quicker execution of the traffic-intensive steps of the computation.

5.3.2 Balanced Coloring

Figure 7 shows the execution times of the Balanced Coloring application on both WiNoC and mesh interconnected multicore systems. These execution times are broken down into the two major phases within computation – viz. initial coloring and redistribution. Among the graphs considered, CAC achieves the maximum improvement in execution time (35%) by using the WiNoC. CAD and CNR follow CAC with 25% execution time improvement. The two street network graphs, BLG and NTH achieve the least execution time improvement (10%).

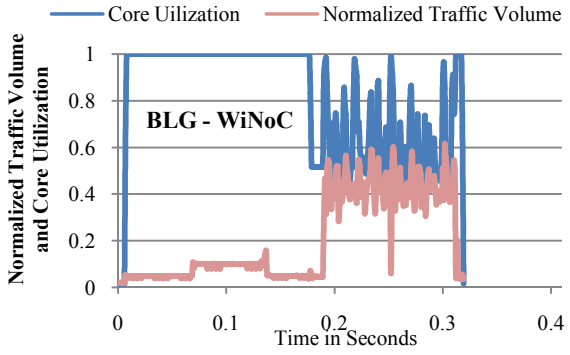


(a) CAD: Redistribution phase with WiNoC

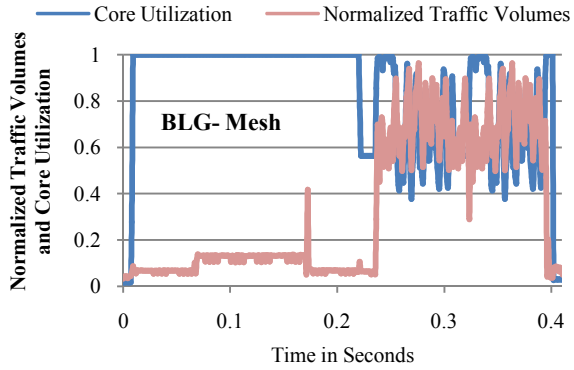


(b) CAD: Redistribution phase with mesh

Figure 9. Balanced Coloring- CAD Runtime Characteristics

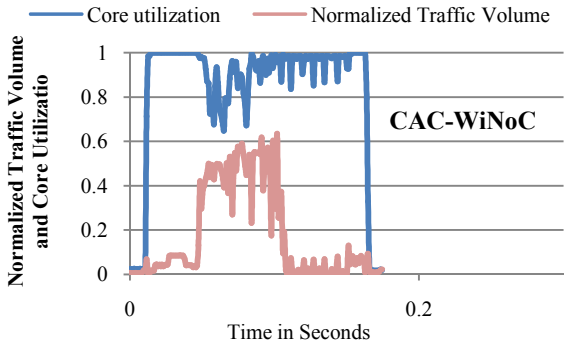


(a) BLG: Redistribution phase with WiNoC

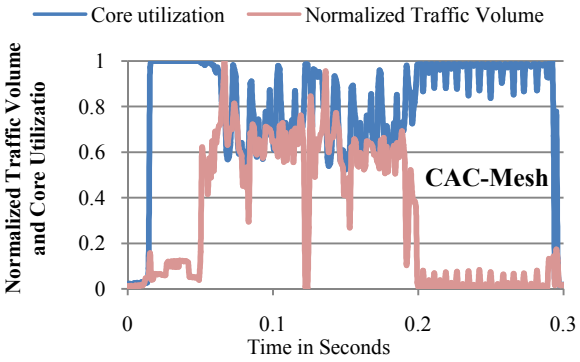


(b) BLG: Redistribution phase with mesh

Figure 10. Balanced Coloring- BLG Runtime Characteristics



(a) CAC: Redistribution phase with WiNoC



(b) CAC: Redistribution phase with mesh

Figure 11. Balanced Coloring- CAC Runtime Characteristics

Figure 8 compares the traffic injection rates of the initial and redistribution phases. The average number of flits injected per cycle per core gives the traffic injection rate. As explained in section 3.2, the redistribution phase involves heavy data migration. Due to their communication intensive nature, the redistribution phases exhibit up to 4x higher injection rates when compared to corresponding initial coloring phases. Hence, redistribution phases achieve higher execution time improvements with WiNoC, when compared to the respective initial coloring phases.

It can be noted from Figure 7 that the graphs with a larger number of colors (>80 colors) achieve a better execution time improvement with WiNoC when compared to graphs with low number of colors (<20 colors). This disparity in the improvement profile can be explained as follows: The graphs with more colors also spend more time in the redistribution phase, which as explained above stands best to gain from the WiNoC architecture. Observing the trends in traffic volumes and CPU utilization from Figures 9, 10 and 11 can corroborate this. More specifically, the CPU activity drops during the redistribution phase when executed with the mesh topology. With higher average hop counts, the mesh topology forces the CPUs to exhibit prolonged waits for necessary data, thereby leading to a drop in CPU activity. In contrast, WiNoC maintains a fairly high CPU utilization throughout the redistribution period.

5.4 System Energy

Figure 12 shows the energy consumptions of the systems executing Grappolo and Balanced Coloring, employing Mesh and WiNoC architectures. The overall energy consumption of the multicore system is the sum of the energies consumed by the cores, interconnects and network switches. As stated earlier, the average hop count of the mesh network is higher than that of the WiNoC. Hence, each injected message stays longer in the mesh network than in the WiNoC network. This in turn leads to higher network energy consumptions. As an example, we can consider the execution of CAD Balanced Coloring on systems with mesh and WiNoC interconnects. By comparing figures 9(a) and 9(b), it can be seen that the average traffic volume associated with mesh is higher than the average traffic volume associated WiNoC.

Compared to the mesh, the WiNoC uses more energy efficient wireless links for long-range communication. This enables WiNoC to achieve a better interconnect energy consumption. In addition, when compared to the mesh, the WiNoC architecture enables a lesser execution time, leading to improved CPU energy consumptions. Due to its communication intensive nature, the Balanced Coloring application achieves a higher improvement in execution time than Grappolo, by using WiNoC when compared to mesh. This in turn, enables the Balanced Coloring application to achieve more pronounced CPU energy savings than Grappolo. More specifically, Balanced Coloring achieves a maximum of 40% reduction in energy consumption by using WiNoC when compared to mesh. Grappolo achieves a maximum of 28% energy savings.

6. Conclusion

Graph analytics have become an essential part of the discovery pipeline in numerous data-driven scientific and social compu-

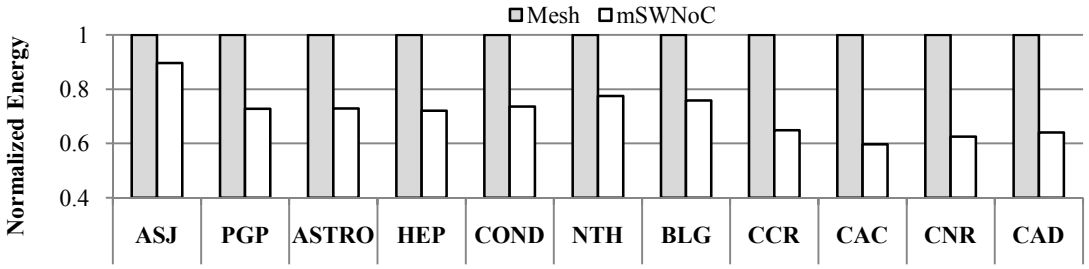


Figure 12. A comparative full system energy profiles for Grappolo and Balanced Coloring, using the mesh and the WiNoC interconnection architectures

ting fields. However, implementing advanced graph operations on state-of-the-art multicore platforms requires significant redesign of the on-chip interconnect topologies to mask the adverse effects of irregular data movement characteristics. In this paper, we analyze the traffic patterns generated by two state-of-the-art parallel implementations for advanced graph analytics – viz. Grappolo for community detection, and Balanced Coloring. The traffic generated by these applications exhibit high long range communication patterns with traffic hotspots and are highly suitable for the implementation of a small world Wireless Network on Chip (WiNoC) architecture. With long-range shortcuts, the WiNoC enables a fast data exchange among the computing cores leading to improved system performance. For the graphs considered, the WiNoC can provide up to 35% execution time improvement and 40% energy consumption improvement in running graph analytics, over the traditional wireline based mesh interconnection network. The study presented in this paper represents, to the best of our knowledge, the first detailed design and evaluation of a WiNoC-based multicore platform for graph applications.

7. Acknowledgements

This work was supported in part by the US National Science Foundation (NSF) grants CCF-0845504, CNS-1059289, CNS-1128624, and CCF-1162202, an Army Research Office grant W911NF-12-1-0373, as well as US DOE award DE-SC-0006516.

8. References

[1] S. Deb, et al., "Wireless NoC as Interconnection Backbone for Multicore Chips: Promises and Challenges", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(2):228-239, 2012.

[2] S. Deb, et al., "Design of an energy efficient CMOS compatible NoC architecture with millimeter-wave wireless interconnects," *IEEE Transactions on Compute*rs, 62(12): 2382-2396, 2013.

[3] E. E. Schadt, et al., "Computational solutions to largescale data management and analysis", *Journal of Nature Reviews Genetics*, 11(9): 647-657, 2010.

[4] S. Fortunato "Community detection in graphs." *Physics Reports,* 486(3): 75-174, 2010.

[5] M. EJ. Newman, "Modularity and community structure in networks." *Proceedings of the National Academy of Sciences,* 103(23): 8577-8582, 2006.

[6] T. R. Jensen, and T. Bjarne, " Graph coloring problems", Vol. 39. *John Wiley & Sons*, 2011.

[7] F. T. Leighton,"A graph coloring algorithm for large scheduling problems." Journal of research of the national bureau of standards, 84(6): 489-506, 1979.

[8] M. T. Jones and P. E. Plassman,"A parallel graph coloring heuristic." *SIAM Journal on Scientific Computing* 14(3):654-669, 1993.

[9] H. Furma´nczyk, Equitable coloring of graphs, in: Graph Colorings, (M. Kubale, ed.), *American Mathematical Society Providence*, Rhode Island , pp. 35–53, 2004.

[10] H. Bodleander and F. Fomin, "Equitable colorings of bounded treewidth graphs," *Theoretical Computer Science*, 349(1):22–30, 2005.

[11] H. Lu, M. Halappanavar, D. Chavarria, A. Gebremedhin, A. Kalyanaraman, "Balanced coloring for parallel computing applications", *Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS),* May 25-29, 2015, Hyderabad, India, Accepted.

[12] D. Ediger, "Analyzing Hybrid Architectures for Massively Parallel Graph Analysis", Ph. D. Dissertation, Georgia Institute of Technology, Georgia, Atlanta, May-2013.

[13] D.A. Bader, G. Cong, and J. Feo, "On the Architectural Requirements for Efficient Execution of Graph Algorithms", in *Proceedings of The 34th International Conference on Parallel Processing (ICPP 2005),* pp. 547-556, 2005.

[14] H. Lu, M. Halappanavar, A. Kalyanaraman, "Parallel Heuristics for Scalable Community Detection", *Parallel Computing*, ISSN 0167-8191, March 2015, doi:10.1016/j.parco.2015.03.003.
http://www.sciencedirect.com/science/article/pii/S0167819115000472

[15] D. Chavarrıa-Miranda, M. Halappanavar, and A. Kalyanaraman, "Scaling graph community detection on the Tilera manycore architecture," in *HiPC 2014*, Goa, India, 2014, 11 pages.

[16] U. Catalyurek, J. Feo, A. H. Gebremedhin, M. Halappanavar, A. Pothen, "Graph coloring algorithms for multi-core and massively multithreaded architectures," *Parallel Computing*, 38:576–594, 2012.

[17] E. J. Riedy, M. Henning, D. Ediger, and D.. Bader. "Parallel community detection for massive graphs." in *Parallel Processing and Applied Mathematics,* pp. 286-296. Springer Berlin Heidelberg, 2012.

[18] C. L. Staudt, and H. Meyerhenke. "Engineering high-performance community detection heuristics for massive graphs." in *Proceedings of 42nd International Conference on Parallel Processing (ICPP)*, pp. 180-189, 2013.

[19] B. Wu, Y. Dong, Q. Ke, and Y. Cai, "A parallel computing model for largegraph mining with MapReduce," in *Proceedings of Seventh International Conference on Natural Computation (ICNC*), pp. 43–47, 2011.

[20] A. Das, et al., "Dynamic Directories: A mechanism for reducing on-chip interconnect power in multicores", in *Proceedings of DATE*, pp. 479-484, 2012.

[21] U.Y. Ogras and R. Marculescu, "It's a small world after all: NoC performance optimization via long-range linkinsertion", *IEEE Trans. On Very Large Scale Integration Systems,* 14(7):693-706, 2006.

[22] Y. Saad, "Iterative Methods for Sparse Linear Systems", *Society for Industrial and Applied Mathematics,* Philadelphia, PA, 2003

[23] T. Petermann, P. Los Rios, "Spatial Small-World Networks: A Wiring Cost Perspective", arXiv: condmat/0501420v2.

[24] P. Wettin, et al., "Energy-efficient multicore chip design through cross-layer approach". In *Proceedings of DATE*, 2013, pp.725-730.

[25] K. Duraisamy, R. Kim, P. Pande, "Enhancing Performance of Wireless NoCs ith Distributed MAC Protocols", in *Proceeedings of ISQED*, 2015.

[26] N. Binkert, et al., "The GEM5 Simulator", *ACM SIGARCH Computer Architecture News*, 39(2):1-7, 2011.

[27] S. Li, et al., "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures", In *Proc., of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469-480, 2009.

[28] DIMACS10, The 10th DIMACS implementation challenge – graph partioning and clustering.

URL: http://www.cc.gatech.edu/dimacs10/

(Last date accessed: June 2015).