Large-Scale On-Chip Dynamic Programming Network Inferences using Moderated Inter-Core Communication

Andrew Mundy, Terrence Mak and Alex Yakovlev
School of Electrical and Electronic Engineering, Newcastle University, Newcastle-upon-Tyne, UK

Simon Davidson and Steve Furber
School of Computer Science, University of Manchester, Manchester, UK

Abstract

The analysis of large scale, complex networks using dynamic programming is of great use in many scientific and engineering disciplines. Current applications often require the analysis of scale-free networks with many millions of nodes and edges; presenting a huge computational challenge. Employing a distributed networks-on-chip infrastructure presents a unique opportunity of delivering power efficient and massive parallel accelerations. However, bursting and asymmetric communications across cores could create instant network saturation and lead to packet loss and performance degradation. In this paper, we present a moderated communication methodology that enables a balanced channel usage and network topological adaptation for improved performance. A novel analytical communication model for NoC is developed and leads to a theoretical bound of the on-chip communication cost estimate. Performances of the many-core computation and the proposed methods are rigorously evaluated using the real 18-core SpiNNaker chip. We demonstrate a 10x speed-up in analysis convergence and a 42% reduction in instantaneous Packet Injection Rate based on benchmark networks.

Index Terms
Graph analysis, Dynamic Programming, Bellman Equation, Dynamic Programming network, Concurrent computation.

1. Introduction

The analysis of large-scale complex graphs, or networks, is of growing importance within the engineering and scientific community[1], [2]. Aside from the direct applications to Networks-on-Chip, such as systems analysis and packet routing; life-sciences, bioinformatics, and the social sciences, are now also frequent users of graph theory and analysis. Uses may be as diverse as transport network planning, to machine learning, to protein interaction analysis in drug discovery, to web-page ranking, and to time and project management. Considered networks may consist of several millions of arcs and vertices, and their processing is consequently costly, often intractable. The topology of such networks may further increase the computational cost associated with their analysis, with issues such as cache-hits, and, when parallelised, process control, potentially proving to further worsen the performance of the given computing equipment.

The computational advantages of multiple processors have been utilised to accelerate network analysis, and with the arrival of multi-core desktop computers, parallel processing is now, to a greater-or-lesser extent, a part of everyday life. Parallel computing, however, may prove costly with respect to programmer effectiveness. Studies have demonstrated the challenges posed to programmers by typical parallel architectures. Issues such as atomicity and memory management can present severe stumbling blocks even to those who are relatively experienced[3], [4]. Whilst platforms such as nVidia’s CUDA can provide some degree of abstraction for users, the monolithic nature of the memory may still pose a problem. Despite this, GPU platforms such as CUDA have been utilised to provide exceptionally fast network analysis, [5]. The power consumption costs of GPU architectures, can, however, be significant[6]. Given worries regarding the sustainability of large-scale computation centres, the use of lower-power alternatives must be considered.
Distributed computing systems, consisting of many distinct, simple, processing cores, each associated with a private memory space, and connected using a Networks-on-Chip architecture, may provide a means by which large-scale complex network analysis may be easily programmed, and cheaply performed[7]. The SpiNNaker computing platform from the University of Manchester, is an example of such a system[8], [9]. The analogue provided by Networks-on-Chip of a large-scale complex network, allows the problem of network analysis to map naturally to the system, with NoC packets representing edges and edge traversal, and processing cores representing vertices. Typically, the number of processors available will be less than the number of nodes within the network, and so some degree of abstraction is required; the use of network clusters on processing cores allows for a complete mapping of network-to-network. Additionally, the use of Dynamic Programming techniques not only provides a means for the programmer to understand operation of the entire system through the operation of a single processing core, but allows for the treatment of each core as a distinct entity, or ‘neuron’[10].

Communication between processing cores is an essential component of the distributed approach to network analysis. Networks-on-Chip present additional limitations on the use of the given communication channel, particularly with respect to packet injection rate. The SpiNNaker system, in particular, is known to have a very resilient Networks-on-Chip architecture[11], though this will not necessarily hold true for all embedded distributed systems. Careful algorithm design is therefore critical, although it is hoped that this will not be overly detrimental to programmer effectiveness. It is necessary, therefore, to develop methods whereby the communications channel may be utilised to full effect without negatively impacting on either system performance, power usage, or programmer effectiveness.

The issues of clustering a network, and mapping it to cores are already well covered; and as such will not be tackled by this paper [12]. The following are provided:

- An analytical method by which the packet injection rate and communication channel usage of an algorithm, or algorithms, may be characterised.
- An evaluation, using the SpiNNaker computing platform, of two representative algorithms. It is found that more frequent message passing results in up to a 10x speed-up of the analysis, although this is a function of network scale and topology, and even larger networks, if better clustered, may result in even larger accelerations.

2. Dynamic Programming Networks

Dynamic Programming (DP) is a powerful means by which a problem consisting of a large set of similar sub-problems may be simply specified and solved. Typical applications may be classified either as those maximising yield within a given time, or those minimising the cost or time required to achieve a result[13]. A generalised form of the Bellman Equation is typically given as the following recursive formulation;

\[
f_i(p) = \min_{vk} \left( f_k(p) + C_{ik}(p) \right)
\]  

Where \(C_{ik}\) is the cost, or reward, associated with transferring from state \(i\) to state \(k\). Enumeration of all possible states yields solution of the problem, though at a significant computational cost – deemed the “curse of dimensionality” [14]. These computational costs may prove prohibitive, particularly when considering the analysis of networks approaching several million nodes and edges.

It can be demonstrated [15] that a dynamic programming system may be transformed to a set of non-linear differential equations,

\[
\frac{df_i(t)}{dt} + \frac{1}{\lambda_i} f_i(t) = \frac{1}{\lambda_i} \min_{\forall k} \{f_k(t) + C_{ik}(t)\}, \forall i
\]  

This characteristic property of the dynamic programming approach allows for real-time computation of the system; whereby convergence is achieved not through iteration of all states, but by allowing the system to minimise towards a result over time. In contrast to classical, “top-down” approaches to system design, where the overall system convergence may be predetermined; use of dynamic programming techniques inherently allows for computation by concurrent agents, each of whom is responsible for the computation of a single component of the system. The system behaviour may be understood as the resultant collective behaviour of its agents; and such a system may be analysed by the means given in [16]. This time based solution has allowed for analogue implementations of DP networks[17].

Whilst analogue implementations and systems may prove to possess exceptionally rapid computational abilities, with very low power consumption; the very
nature of the implementation as a physical device reduces effective scalability. Digital systems, in contrast, provide a means for easy reprogramming and network definition; and are fully capable of displaying the energy minimising, collective behaviour exhibited by this class of problem [10], [18]. Alongside the form of analysis demonstrated here, the Bellman Equation may be altered to provide other forms of network analysis. Not only does this mean that a single implementation of the digital dynamic programming network may be simply modified to calculate other network metrics; but also that the overall convergence properties may be foreseen.

The communications channel provided by NoC architectures is of the utmost utility when considering the development of digital DP networks. Assuming that each processing core may be considered as a neuron-like entity, responding to input as required, and stimulating the operation of other cores as and when required, the network acts as a fabric facilitating simple communication between processing cores. The Network-on-Chip may thus act in a manner similar to the edges within the network to be analysed, with nodes performing the requisite processing. The network forms a critical stage of the computation process, as shown below, and the loss or delay of packets will not only severely hamper convergence time, but may lead to significantly erroneous results.

3. Methodology

Consider a scale-free graph of the form $G(V, E)$ consisting of nodes, $V$, and edges, $E$. This graph is to be partitioned, or clustered, to allow for computation using $M$, memory limited, processing cores. Thus, it shall be stated that $G$ is partitioned to produce $G_1(V_1, E_1), G_2(V_2, E_2), \ldots, G_M(V_M, E_M)$. Whilst all graph vertices shall be assigned a subgraph, such that $V = V_1 \cup V_2 \cup \cdots \cup V_M$, not all edges may be assigned thus. Edges where this is the case, such that $u_e \in V_n$ and $v_e \not\in V_n$, shall be termed intercluster, and represented with the set $E_O$. In addition, each subgraph shall also contain a set with incoming intercluster edges, $E_i$, and a set with outgoing intercluster edges, $E_o$, such that $G_n(V_n, E_n, E_{on}, E_{in})$ and $E_{in} \cup E_{on} \subseteq E_O$. Additionally, let $u_e$ and $v_e$ represent the originating and terminating nodes of edge $e$, respectively, and $v_u$ represent the node prior to $v$ in the shortest path from a given originating node.

Two edge cases may now be considered: when $M = 1$, and when $M \geq |V|$. In regard to the former, no intercluster edges may be in existence, and thus all computation is to be performed upon one core; as such, there is no communicational cost to consider. In contrast, should one vertex be assigned to each computational core, the computational cost would be negligible, and the communicational cost would dominate the performance of the system. Between these points we expect the computational cost to decrease asymptotically to zero with increased core count (as in Amdahl’s Law[19]), but for the communicational cost to increase.

Edge traversal is now to be achieved by two means: within the subgraph stored upon a processing core, traversal is hidden by the operation of the given analysis algorithm; intercluster edges, must, however, be traversed by means of message passing between processing cores. A communicational cost is, therefore, to be considered when both designing and analysing algorithms. Two extreme cases of communication may be envisaged, as in Figure 1: a spiking, high intensity, low-packet system; and a system with higher numbers of dispatched packets, but a lower instantaneous packet injection rate.
injection rate. As regards communication complexity, the former is desired due to the lower number of dispatched packets; the latter, however, may represent less overall cost due to reduced packet injection rate.

The primacy of network analysis within current scientific research has been noted above. Whilst several forms of network analysis are both useful, and amenable to implementation using DP techniques, the Single-Source Shortest Path Problem has been selected as an illustrative example. Future work will consider the use of distributed NoC systems for other forms of network analysis. The Bellman-Ford Algorithm, Figure 2, is a dynamic programming formulation of the Single-Source Shortest Path Problem[20], and has thus been adapted for use here.

A network-enabled form of the Bellman-Ford Algorithm is shown in Figure 3, and has complexity $O(|V||E| + |E_{on}|)$. The algorithm has been modified to transmit packets to traverse intercluster edges after the main loop of the Bellman-Ford algorithm; and to only traverse an edge if the originating node has been relaxed since the last time the edge was traversed. The previous notation of $\pi(u)$ representing the distance of the node $u$ from a given starting node has been extended such that $\pi^t(u)$ represents the distance at discrete time $t$. External edge traversal is managed by means of packet transmission (the function TRANSMITPACKET in the given algorithms); each packet consists of a routing key, and a 32-bit payload split to contain the unique ID of the intercluster edge and the new distance of the originating node. Upon receipt of such a packet, a node will scan the incoming edge list, $E_{in}$, to determine if the edge terminates within the given subgraph, should this be the case then relaxation of the terminating node and further computations may be attempted.

[21], [22] demonstrate that the convergence time of distributed algorithms may be reduced by increased message passing, and thus communication complexity. Figure 4 shows a proposed algorithm that allows for edge traversal with greater frequency than the network-enabled form of the Bellman-Ford algorithm, although a worsened computational complexity of $O(|V||E| + |E_{on}|)$ may be demonstrated.

Increased communication complexity is generally avoided to ensure adequate network performance. The relationship between hardware and software, particularly with regard to NoC, is, however, more complex than is generally recognised. [23] The analysis performed here on system communication channel usage attempts to recognise the very specific links between problem topology, algorithm construction, and packet-loss by means of injection rate.

**HYBRIDIEF-BELLMANNFORD**($G_n(V_n, E_n, E_{on})$

1  for $i = 0$ to $V_n$.length
2     for $e \in E_n$
3       if $\pi^t(v_e) > \pi^t(u_e) + |e|$
4           $\pi^t(v_e) = \pi^t(u_e) + |e|$
5           $v_u = u_e$
6    for $e \in E_{on}$
7       if $\pi^{t-1}(u_e) > \pi^t(u_e)$
8          TRANSMITPACKET($e$)
9          $\pi^{t-1}(u_e) = \pi^t(u_e)$

Figure 4. A communicationally moderated implementation of the Bellman-Ford algorithm, entitled the Hybrid or HybridIEF (Iterative/Edge-Following) approach. External edges are traversed at the same point within the iteration as internal edges; accelerating the overall algorithm convergence, and reducing instantaneous packet injection rate.

**ITERATIVE-BELLMANNFORD**($G_n(V_n, E_n)$

1  for $i = 0$ to $V_n$.length
2     for $e \in E_n$
3       if $\pi^t(v_e) > \pi^t(u_e) + |e|$
4           $\pi^t(v_e) = \pi^t(u_e) + |e|$
5           $v_u = u_e$

Figure 2. The iterative form of the Bellman-Ford algorithm.

**NETWORKED-BELLMANNFORD**($G_n(V_n, E_n, E_{on})$

1  ITERATIVE-BELLMANNFORD($G_n(V_n, E_n)$
2     for $e \in E_{on}$
3       if $\pi^{t-1}(u_e) > \pi^t(u_e)$
4          TRANSMITPACKET($e$)
5          $\pi^{t-1}(u_e) = \pi^t(u_e)$

Figure 3. The iterative form of the Bellman-Ford algorithm may be easily adjusted to allow for external edge traversal. In this networked implementation, external traversal is performed after the main portion of the Bellman-Ford algorithm has been performed.
3.1. Analysis of Communications Network Usage

If, as in the given algorithmic implementations, to reduce communicational channel usage, an intercluster edge is only to be traversed when the distance of the originating node is decreased, an expression for the number of times an external edge will be traversed, and thus a packet sent, within a discrete time interval may be written as,

\[ p(t_0, t_1, e) = \begin{cases} 1, & \pi^t(u_e) < \pi^{t_0}(u_e) \\ 0, & \pi^t(u_e) \geq \pi^{t_0}(u_e) \end{cases} \]

(3)

for \( t_1 > t_0 \) and \( e \in E_{\text{on}} \). It is therefore assumed, that edge relaxation is atomic within the constraints of the discrete time model.

This notation may be extended such that, for one run of the Networked (Iterative/I) Bellman-Ford algorithm, lasting discrete time \( \tau \), the total number of packets sent, and hence external edges traversed, may be expressed as:

\[ P^i_\tau = \sum_{\forall E_{\text{on}}} p(0, \tau, e) \]

(4)

Similar application to the Hybrid algorithm results in the following,

\[ P^h_\tau = \sum_{t=1}^{\tau} \sum_{\forall E_{\text{on}}} p(t-1, t, e) \]

(5)

For the same graph, it may be assumed that the hybrid algorithm will result in a greater number of packets being sent; as, for a single intercluster edge,

\[ \sum_{t=1}^{\tau} p(t-1, t, e) \geq p(0, \tau, e) \]

(6)

Whilst, however, it is important to minimise communicational complexity, and thus the number of packets sent, the likelihood of a packet being dropped is most closely proportional to packet injection rate[15, Figure 8]. Therefore, any analysis or comparison of algorithms must also account for packet injection rate. Simple expansion of the above notation yields the average packet injection rate of the two algorithms over one entire run of the algorithm; this, does not, however, accurately reflect the operation of each algorithm (in as much as neither algorithm traverses external edges at a single determinable rate). Instead, let \( k \cdot |E_{\text{on}}| \) represent a bounded approximation of the time required to scan through the outgoing intercluster edge list and determine which edges require traversal. Figure 5 demonstrates the modelled operation of the two algorithms, with respect to packet injection rate.

Given that the Iterative algorithm scans the outgoing edge list and transmits packets once per run, it may be stated that,

\[ \hat{r}^i = \frac{P^i_\tau}{k \cdot |E_{\text{on}}|} \]

(7)

where \( \hat{r}^i \) expresses the peak packet injection rate for the iterative algorithm.

In contrast, the Hybrid algorithm scans the list once per iteration of the outermost loop, and will thus send packets \( |V| \) times for one run of the algorithm. As it is not possible, without a given network, to state the number of packets that will be sent at any stage within this process, the peak packet injection rate can be approximated thus,

\[ \langle r^h \rangle = \frac{P^h_\tau}{k \cdot |V| \cdot |E_{\text{on}}|} \]

(8)

A ratio of the peak packet injection rates of the algorithms may be expressed as,

\[ \frac{r^i}{\langle r^h \rangle} = \frac{|V| \cdot \sum_{\forall E_{\text{on}}} p(0, \tau, e)}{\sum_{\forall E_{\text{on}}} \sum_{t=1}^{\tau-1} p(t-1, t, e)} \]

(9)

The worst case communication channel usage exists when the originating vertex of each outgoing edge is relaxed at least once within time \( \tau \), and thus, it can be stated,

\[ \frac{r^i}{\langle r^h \rangle} = \frac{|V| \cdot |E_{\text{on}}|}{k \cdot |E_{\text{on}}|} = |V| \]

(10)

Figure 5. Modelled Network Utilisation for One Processing Core. The operation of the iterative algorithm (dotted impulse) and the operation of the hybrid algorithm (dashed impulses) are shown. The hybrid algorithm is much nearer in channel usage characteristics to the moderated scheme shown above.
Where $\chi$ is a function of network topology and represents the ratio of the number of nodes relaxed only once to the number of nodes relaxed multiple times. By solely considering networks that are scale free in nature, it can be shown that the number of nodes that will have their distance reduced $n$ times, decreases with $n$, following the power-law; and thus stated that $\chi = f(\sigma)$. More specifically, given a discrete power law of the form $\rho(n) = an^{\alpha\sigma} + \beta$, where $\rho(n)$ is the number of nodes relaxed $n$ times during one run of the algorithm, and $\hat{n}$ is the peak number of subsequent relaxations experienced by any node in the system,

$$
\chi = \frac{\hat{n}}{\sum_{n=2}^{\hat{n}} an^{\alpha\sigma} + \beta} = \frac{\hat{n}}{(\hat{n} - 1)\beta + a \sum_{n=2}^{\hat{n}} n^{\alpha\sigma}}
$$

(11)

**Summary:** The cumulative packet injection for both the “Iterative” (4) and “Hybrid” (5) algorithms has been derived; and it has been shown that the hybrid algorithm transmits a greater or equal number of packets than the iterative algorithm (6). However, by inspecting peak packet injection rates, a ratio has been derived (9)(10), which demonstrates the theoretical superiority of the hybrid algorithm in the case of scale-free network topology (11). By careful inspection, therefore, of network topology, the appropriate algorithm for the network in question may be chosen; though in the majority of cases we expect $|V| \chi > 1$, and thus for the Hybrid Algorithm to provide improved performance with respect to network utilisation.

**4. Results**

To demonstrate the operational differences and validity of the proposed algorithms, a series of test networks were generated. The benchmark tests generated were inspired by the SSCA benchmarks for High Productivity Computing Systems[24]. To reduce the need to assess algorithms capable of partitioning or clustering a graph, subgraphs were initially generated for each processing core and then joined by intercluster edges during a merger process. Each generated subgraph was of equal size and fitted a scale-free distribution; and intercluster edge generation attempted to preserve this topology.

**4.1. Implementation on SpiNNaker**

Each SpiNNaker chip contains 18 processing cores, a maximum of 17 of which may be used by the programmer at any time. Each core has access to a very limited amount of memory; and message passing between cores is achieved by passing a “spike” to the fabric of the NoC. Each spike is assigned a routing key, by which it is routed to other cores present on the chip, or to the toroidal mesh that connects the multiple chips present within a SpiNNaker system. Each spike may also carry a 32-bit payload, and it is this that is used to provide a means of edge traversal. On receipt of a spike by a processing core, a callback function is called to determine the actions that are to be undertaken: they to update the state of a neuron; or to perform, if required, network analysis of the subgraph assigned to the core.

Using the provided C API, a system for performing network analysis on SpiNNaker was created. Networks may be specified to the processing core by use of the SpiNNaker datagram protocol, and timing results read back by the same means. Callback handlers are used to respond to incoming edge packets, and schedule operation of the analysis algorithm if required. On receipt of such a packet, the list of incoming edges is scanned, and relaxation is attempted if an edge match is found. If the terminating node may be relaxed then the specific analysis algorithm is scheduled to run (if not already scheduled), otherwise no action is taken. To allow for timing, the start time on each core is recorded, and each algorithm saves the simulation time to a variable upon completion; upon receiving a request for the timing information, a core reports a start and stop time, which may be used to determine the overall convergence time.

Once the SpiNNaker system has been initiated, by means of flashing the compiled executable onto each processing core, network analysis may be performed. Using a communications shell developed as part of

![Figure 6. A benchmark network, for 16 processing cores, generated by combining scale-free subgraphs and inserting intercluster edges.](image-url)
the project, a network may be analysed by loading in the network and clustering data files; propagating (‘seeding’) that data to the requisite cores; selecting the originating node; and starting the selected algorithm. Results may then be retrieved, along with timing results, if so required. A maximum network size of 2000 nodes, 4000 edges, 50 incoming external edges, and 50 outgoing external edges may be stored in the memory of each core. This may be tailored as required, to increase, for example, the number of intercluster edges.

4.2. Core Count Related Acceleration

To demonstrate the acceleration caused by utilising several processing cores, a series of networks were generated that, when combined, would fit within the network storage space of one processing core; these networks possessed around 2000 nodes and 3000 edges; and form the basis of the test benchmarks, mentioned above. These allowed for demonstration of the convergence time when the same network was distributed to differing numbers of processing cores. Figure 7 shows the average convergence time of all three benchmark networks undergoing this process.

It is important, here, to note that the super-linearity demonstrated by the convergence acceleration may be explained. Given a network \( G(V, E) \) and an analysis algorithm of complexity \( O(|V| \cdot |E|) \), the work performed by analysis of the network on one core is \( O(|V| \cdot |E|) \). If the network is then clustered to provide two smaller subgraphs, \( G_1 \) and \( G_2 \), where \( |V_n| \approx \frac{|V|}{2} \) and \( |E_n| \approx \frac{|E|}{2} \); the computational work to be performed by each core is now:

\[
p_n = O((|V_n| \cdot |E_n|)) \approx O \left( \frac{|V| \cdot |E|}{4} \right)
\]

Acceleration is performed by attempting to ensure that operation of the two cores is concurrent; by means of frequent message passing; and by substituting some degree of computational complexity by communication complexity. The ‘Bound’ plotted in Figure 7 is a work-scaled inverse of the core-count and represents the maximum acceleration possible without exhibiting double-super-linearity.

Not only can it be seen that for lower core-counts, the Hybrid algorithm demonstrates a significantly faster rate of convergence; but also that the communicational saturation predicted above, and linked to Amdahl’s Law, leads to reduced acceleration for successively greater core counts. An 8.60x acceleration is achieved by using the iterative algorithm and 10 processing cores, whilst a 10.65x acceleration can be achieved through use of the hybrid algorithm and 7 processing cores.

Errors for these results, and all subsequent results, stem from several factors inherent within SpiNNaker, and the network analysis system. Primarily, the asynchronous nature of the SpiNNaker system makes it necessary to provide convergence timers on all processing cores; these record the approximate time of the start of the algorithm, and record a potential completion time after each run of the algorithm. The sources of error, thus, are in the delay between packets starting the timers on each core; and the maximum possible error is a function of the number of cores utilised in the computation.

4.3. Iterative vs. Hybrid

With respect to the previous results, it may be posited that an optimal number of cores may be selected for any computation to ensure the fastest convergence. Figures 8a and 8b demonstrate the convergence times for various network sizes and core counts. It may be seen that the rate of growth of convergence time is proportional to core count; and represents the growing computational and synchronisation cost posed by the growing network scale. The communicational costs are such that two cores may provide faster convergence than sixteen cores for smaller networks.

Additional comparisons between the suggested algorithms may also be made. Figure 9 demonstrates the acceleration provided by the Hybrid Algorithm.
with respect to the Iterative Algorithm. The send-early approach of the hybrid algorithm yields greater acceleration in cases of higher core counts, mostly due to the greater level of communication required. A maximum acceleration of 16% of the hybrid algorithm in regard to the iterative algorithm may be observed from our test results; although the indication is that acceleration is a function of network size.

4.4. Network Utilisation

The benchmark networks, §4.2, were utilised to measure the peak packet injection rate of a single core performing the initial analysis of the given network. Table 1 shows the injection rates measured, along with the improvement achieved by use of the Hybrid algorithm. It is important to note that all these networks are scale free; and that the improvement demonstrated is a function of network topology, as mentioned above. A well designed clustering algorithm should also be able to maximise the gains made available by the Hybrid algorithm. Figure 10 shows captured network usage from analysis of a benchmark network; the predicted higher instantaneous injection rate of the iterative algorithm may be observed.
Table 1. Measured Peak Packet Injection Rate, for Benchmark Networks of 2000 nodes and 3000 edges

<table>
<thead>
<tr>
<th>Number of Cores in Calculation</th>
<th>Benchmark Network</th>
<th>Iterative Algorithm</th>
<th>Hybrid Algorithm</th>
<th>Reduction / %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A</td>
<td>300</td>
<td>180</td>
<td>40.00</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>300</td>
<td>200</td>
<td>33.33</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>340</td>
<td>320</td>
<td>5.88</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>380</td>
<td>300</td>
<td>21.05</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>280</td>
<td>160</td>
<td>42.86</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>260</td>
<td>220</td>
<td>15.38</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>340</td>
<td>320</td>
<td>5.88</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>200</td>
<td>200</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>280</td>
<td>240</td>
<td>14.29</td>
</tr>
<tr>
<td>16</td>
<td>A</td>
<td>240</td>
<td>220</td>
<td>8.33</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>120</td>
<td>120</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>240</td>
<td>220</td>
<td>8.33</td>
</tr>
</tbody>
</table>

5. Discussion

Mention has been made of the importance of clustering within the workflow of analysing a large-scale complex network using a networks-on-networks approach; particularly with regard to ensuring appropriate workload balancing, external edge connections, and memory utilisation. Mapping and routing are similarly crucial for very large scale networks. As the scale of target network increases, the challenge of appropriately routing edge traversal packets throughout the system increases significantly. Schemes such as Region-Based Routing[25], or Logic Based Distributed Routing[26] are particularly suitable for the scale-free networks discussed. In such a scheme, tightly connected subgraphs may be mapped closely together to ensure a high inter-core bandwidth, and more loosely coupled subgraphs may be distributed with greater geographic spacing. Additionally, the routing system may abstract the programmer further from the challenges of ensuring adequate channel utilisation.

Alternative topologies of network structure may also yield greater networked performance. The ongoing work on 3D NoC, promises significantly greater channel capacity between processing cores. The 3D design paradigm will also allow for network packing with greater geometric density, and should techniques such as FARM[27] prove suitable for 3D devices, the communicational capabilities should be significant.

6. Conclusion

The ability of a distributed computing system, such as SpiNNaker, to perform network analysis has been demonstrated; as have the time, and network utilisation, benefits of utilising a ‘send-early’ algorithm design. Performance, thus far, is not comparable to GPU based solutions for network analysis; though this is an unfair comparison when considering the relative number of cores and clock speeds available. Considering a SpiNNaker ‘103’ system, comprising 68 usable processing cores, we expect to be able to provide analysis of networks consisting of up to 136,000 nodes and 272,000 edges; additionally, we envisage that for ever-larger networks, distributed systems will begin to outperform alternative approaches. When considering the aforementioned programmer related costs, experience with the SpiNNaker system has demonstrated the ease with which a network analysis system may be built and tested. Future work will consider the cumulative packet injection rates of the given algorithms, the design of other analysis algorithms, and will assess clustering algorithms for use with the SpiNNaker network analysis system.

Acknowledgments

We gratefully acknowledge the help and support of Dr Steve Temple of the School of Computer Science, University of Manchester, during the period of software development for SpiNNaker.

References


