Energy Efficient MapReduce with VFI-enabled Multicore Platforms

Karthi Duraisamy*, Ryan Gary Kim†, Wonje Choi*, Guangshuo Liu†, Partha Pratim Pande*, Radu Marculescu†, Diana Marculescu†

*School of EECS  
Washington State University  
Pullman, WA 99164, U.S.A.  
{kdurais, rkim, wchoi1, pande}@eecs.wsu.edu

†ECE Department  
Carnegie Mellon University  
Pittsburgh, PA 15213, U.S.A.  
{gsliu, radum, dianam}@cmu.edu

ABSTRACT

In an era when power constraints and data movement are proving to be significant barriers for high-end computing, multicore architectures offer a low-power and highly scalable platform suitable for both data- and compute-intensive applications. MapReduce is a popular framework to facilitate the management and development of big-data workloads. In this work, we demonstrate that by using a wireless NoC-enabled Voltage Frequency Island (VFI)-based multicore platform it is possible to enhance the energy efficiency of MapReduce implementations without paying significant execution time penalties. Our experimental results show that for the benchmarks considered, the designed VFI system can achieve an average of 33.7% energy-delay product (EDP) savings over the standard baseline non-VFI mesh-based system while paying a maximum of 3.22% execution time penalty.

Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design

General Terms

Algorithms, Performance, Design.

Keywords

NoC, Big data, Wireless, Multicore, VFI, low power.

1. Introduction

There is great interest in designing multicore chips that are customized for emerging big-data workloads [1], [2], [3]. In this regard, energy-efficient implementations of the MapReduce framework using single chip multicore platforms can enable new discoveries in big data computing. Due to the widespread use of the MapReduce framework, it is imperative to investigate energy efficient multicore chip design targeted for this application.

Multiple Voltage Frequency Island (VFI)-based designs are common for designing low-power multicore chips for both embedded and high performance multicore platforms [4], [5], [6], [7]. The basic principle of a VFI-based design is that the entire chip is divided into multiple clusters depending on the computation and communication requirements of the cores [4]. Each cluster is then assigned a suitable voltage/frequency (V/F) pair according to the Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

DAC ’15, June 07 – 11, 2015, San Francisco, CA, USA  
Copyright 2015 ACM 978-1-4503-3520-1/15/06…$15.00  
http://dx.doi.org/10.1145/2744769.2744835

cluster’s computation and communication characteristics. By fine-tuning the V/F pairs of each cluster it is possible to achieve significant energy savings within a certain performance constraint.

As the execution of MapReduce on a multicore platform generates varying workload patterns depending on the execution stages, we can accordingly map the cores into several V/F domains. Hence, VFI-enabled multicore design is capable for implementing energy efficient MapReduce. Now, most of the existing VFI-partitioned designs use the conventional multi-hop mesh-based Network-on-Chip (NoC) architecture. For large-scale systems, the inter-VFI data exchanges through traditional mesh NoCs introduce unnecessary latency and energy overheads. Consequently, finding new architectures that can significantly improve the energy consumption and data rates of massively integrated multicore platforms is of great interest.

In this work, we propose the design of a wireless NoC (WiNoC)-enabled VFI-based multicore chip, tailor-made for MapReduce applications. Prior work has already identified the great potential of using long-range wireless links for energy-efficient multicore platforms [8]. We demonstrate that a WiNoC-enabled VFI system is capable of improving the energy efficiency of MapReduce without incurring noticeable performance degradation compared to a conventional mesh-based NoC. The achievable energy savings and the potential execution time penalties depend on the specific MapReduce application. For each application considered, WiNoC-enabled VFI outperforms the corresponding mesh-based design.

2. Related Work

MapReduce is a popular and efficient framework to implement big data computing on large-scale clusters [9]. We consider the Phoenix++ [10] MapReduce framework for the design and evaluation of the VFI-enabled multicore platform presented in this work. Possibilities of reducing on-chip interconnect energy for the Phoenix++ applications through the use of dynamic directories are presented in [11]. Dynamic directories can reduce the exchange of network control messages, leading to energy conservation. The use of a 3D-stacked memory-logic system for MapReduce workloads is explored in [3]. Such designs involve the use of two separate specialized architectures for Map and Reduce phases.

Multicore chip implemented with multiple VFIs is a well-known design paradigm to achieve energy efficiency [12]. Conventional NoCs use multi-hop, packet-switched communication. Design and

![Figure 1: Phoenix++ MapReduce Execution flow](image-url)
optimization of multi- and many-core systems-on-chip (SoCs) that exploit small-world effects have already been demonstrated in [13]. A comprehensive survey regarding various WiNoC architectures is presented in [14], which shows the possibility of creating novel architectures by inserting on-chip wireless links. Although exploiting the small-world effects has been initially used to improve the multicore performance [15], it has been later demonstrated that small-worldness can also benefit the power management via control-theoretic approaches [16].

It is shown in [4] that a WiNoC improves the energy dissipation and latency of VFI-enabled multicore chips in presence of SPLASH-2 and PARSEC benchmarks. However, [4] only characterizes the NoC without addressing the core level issues.

In this work, we explore the opportunities for energy reduction on both processing cores and interconnection network to design an energy efficient multicore platform for MapReduce applications. In this effort we incorporate VFI and the emerging wireless NoC paradigm in a single platform to improve the energy dissipation profile with minimal performance impact when compared to traditional mesh architectures.

3. Introduction to MapReduce

3.1 Computation Pattern

MapReduce is a simple and efficient programming model that targets data intensive computations [9]. These applications involve two distinct execution phases, Map and Reduce. Each MapReduce application is associated with a set of characteristic parameters, called keys and values. Typically, these applications operate on a set of data and identify the values that are associated with each key [17]. To elaborate on this, we consider one of the MapReduce applications, Word Count. This application counts the occurrence of each unique word in a text document. In this case, each unique word can be considered as a key. The total count of each unique word is the value that is associated with it.

Phoenix++ [10] is a p-thread implementation of the MapReduce programming model for multiprocessor systems. The Phoenix++ implementation can be split up into four general execution stages, Split, Map, Reduce and Merge. The execution stages and the data flow paths associated with the Phoenix++ MapReduce framework are shown in Figure 1.

During the Split phase, the input data is divided into multiple similarly sized sub-units. The number of available cores and the nature of the application determine the number of data units created. The Phoenix++ scheduler then assigns each data unit to one of the cores that performs the Map operation. During the Map phase, each core processes all assigned data units and generates intermediate (key, value) pairs [17]. During the Reduce and Merge execution phases, all values associated with each unique key are processed and combined to generate the final output [17].

For most of the Phoenix++ applications, the execution of the Map phase is significantly longer than the Reduce and Merge phases combined. This is due to the fact that the Map phase involves intensive computation on large amounts of data that are transferred from main memory to the processing cores. Compared to the Map phase, the Reduce and Merge phases are less computationally intensive. However, the intensity of the inter-core communication in the Reduce and Merge phases is high due to the transfers of intermediate Keys and Values.

3.2 Task Stealing

Phoenix++ uses a task stealing mechanism to enhance the execution efficiency of MapReduce on multicore systems [17]. At the beginning of each MapReduce phase, the Phoenix scheduler divides the data into multiple sub-units. The computation associated with each data unit is labeled by the scheduler as a ‘task’. The nature of the application, execution phase and the number of available computing cores determine the number of tasks created. Once created, each task is assigned to one of the processing cores. Once a core completes all the tasks assigned to it, it starts to steal the unfinished tasks that are initially assigned to other cores in order to speedup overall execution. This task stealing modifies the computation profiles of the cores in a VFI system when compared to a non-VFI system. As we explain in a later sub-section, for VFI-enabled systems, the pattern of task stealing needs to be modified in order to minimize the variation in the overall computation footprint.

4. VFI Creation

In this section, we describe how we create various VFI clusters in a multicore chip. Then, we discuss the additional analysis needed to fine-tune the VFIs for Phoenix++ applications. This includes adjusting the V/F of certain clusters to accommodate bottleneck cores that exist in Phoenix++ applications and task stealing modifications to account for the VFI creation.

4.1 VFI Clustering

An ideal VFI-based clustering encapsulates most significant inter-core communication within VFIs so that the overall network latency can be improved. It is also expected to group cores of similar utilization into the same cluster so that per-island voltage/frequency scaling can be more effective. Therefore, when implementing VFI-based clustering, we take both processor utilization and inter-processor communication into account. The baseline architecture consists of n homogeneous cores, which are clustered into m equally sized VFIs. In this work, we consider 64 homogeneous cores clustered into four 4×4 equally sized VFIs.

We start by introducing some clustering notations. Per-core utilization is denoted by $u \in [0, 1]^n$, which is defined by the number of instructions committed on average in each cycle normalized with respect to the issue width of the core. The traffic data between every source-destination pair, $f_{ij}$, denotes the number of packets sent from core i to core j per unit time. We also use $X_{ij}$ to indicate whether or not a core is assigned to a cluster, namely $X_{ij} = 1$ if core i is assigned to cluster j, else $X_{ij} = 0$. Since $f_{ij}$ and $u$ vary for different benchmarks, the creation of VFIs is benchmark specific. We define the combined objective of cost as:

$$
\min \left\{ \omega_c \sum_{i,j,p,q} X_{ij} X_{pq} f_{ij} \varphi_{comm}(j, q) + \omega_u \sum_i X_{ij} (u_i - \bar{u}_j)^2 \right\}
$$

subject to

$$
\forall i: \sum_j X_{ij} = 1 \quad \text{and} \quad \forall j: \sum_i X_{ij} = n/m, \text{ where } X_{ij} \in \{0, 1\}
$$

where $\varphi_{comm}$ is the normalized inter-cluster communication cost function, defined as:

$$
\varphi_{comm}(j, q) = \begin{cases} 
1 & \text{if } j \neq q \\
1 / \sqrt{m} & \text{if } j = q 
\end{cases}
$$

$\bar{u}_j$ denotes the mean in each m-quantile of the utilization values. Intuitively, the first half of the objective evaluates total communication cost and the later half evaluates the utilization cost via inter-cluster core utilization variation. $\omega_c$ and $\omega_u$ are the weights for communication and utilization costs, respectively.

The communication cost function $\varphi_{comm}$ assumes that intra-cluster communication cost is evaluated as $1 / \sqrt{m}$ of inter-cluster communication cost. This is because, on average, the number of hops required for inter- vs. intra-cluster communication comes as a ratio of $\sqrt{m}$:1 if an m-cluster (i.e., m equal partitions) implementation is assumed and the m clusters form a square grid. The parameters $\omega_c$ and $\omega_u$ can be arbitrary real numbers and only
the relative magnitude of these two parameters matters. Indeed, if \( \omega_e \) is significantly greater than \( \omega_{sr} \), the solution will be very effective on grouping significant inter-core communication within clusters. On the contrary, if \( \omega_e \) is significantly less than \( \omega_{sr} \), the optimization approaches minimize variation of intra-cluster core utilization. In our setup, we normalize the elements in \( f \) and \( u \) with respect to their maximum values and choose \( \omega_e = \omega_{sr} = 1 \). The objective in (1) is a quadratic function in \( X \); the problem is therefore a 0-1 quadratic programming which is NP-hard and needs to be solved using a branch-and-bound approach since the variables are all discrete. We use a commercial solver, Gurobi [18], to solve it.

### 4.2 Modifying V/F Assignment

We have observed in certain Phoenix++ applications that the core utilization pattern is mostly homogeneous among all cores, except a few bottleneck cores that are more active than others. With this type of nearly homogeneous core utilization, depending on the impact of the traffic patterns, the bottleneck cores may fall within a cluster with a lower than required V/F leading to a high execution time penalty. The lower V/F values will slow down the bottleneck cores leading to slowing down the execution of entire application.

This effect is more prevalent in applications with longer library initialization periods and longer merge periods. Library initialization occurs once before each Map phase. Library initialization function is responsible for initial task scheduling and storage space allocation for Keys and Values. The master cores are responsible for invoking library initialization and hence they tend to be more active than the other cores in the above-mentioned period. Activity in the Merge phase can be understood from Fig. 1. Merge phase involves multiple sub stages and the numbers of active threads gets reduced with each sub stage. Thus in Merge phase specific cores are expected to be more active than the others.

As an example, Fig. 2 presents the distribution of the core utilization values for four Phoenix++ applications, namely Kmeans, Principal Component Analysis (PCA), Matrix Multiplication (MM) and Histogram (HIST), while running on a 64-core system. The input data sizes for these applications are provided later in Table 1. All four applications have notable library initialization periods and indicate the presence of bottleneck cores whose utilization values are higher than other cores. Furthermore PCA has a long Merge period contributing to the bottleneck core effect. In the case of Kmeans, along with bottleneck core effect, the core utilization values vary largely over the 64 cores. Kmeans involve multiple MapReduce iterations (two MapReduce iterations for the data set we considered, with each iteration having all the execution stages listed in Fig. 1) and performs a partitioning operation on the given set of data (usually partitions a large set of numbers into multiple groups). By the nature of the Kmeans application, fewer cores are expected to be more active in the second MapReduce stage as the data partitioned in various groups start to achieve convergence. The non-homogenous core utilization pattern of Kmeans ensures that the bottleneck cores are placed in the V/F clusters with high V/F values. For PCA, MM and HIST, the core utilization pattern is nearly homogeneous and this can lead to suboptimal V/F allocation for the bottleneck cores, for the reasons discussed above. As the number of bottleneck cores is much less than the size of the cluster, creating a separate cluster for the bottleneck cores is not beneficial due to the associated overhead. Moreover, creating a specific cluster for the bottleneck cores will deviate from our baseline uniform VFI clustering discussed in Section 4.1.

To address the execution time penalty created by the slowing down of bottleneck cores, a reassignment of V/F for the designed clusters is carried out specifically for PCA, Matrix Multiplication and Histogram. During the V/F reassignment, clusters in which the bottleneck cores reside are assigned with higher V/F values, while the V/F values of the other clusters remain unchanged. The locations of the bottleneck cores are not shifted to another cluster so that the traffic patterns remain unchanged. Among the other MapReduce applications considered in this work, Linear Regression has very little library initialization period and does not have any merge period. Word Count has a non-homogeneous core utilization pattern like Kmeans. Hence, for these two applications also we do not require any V/F reassignment.

### 4.3 Modifying Task stealing

The necessity for modifying the default task stealing mechanism can be understood from the following case study of the Word Count application. For the Word Count application, among the four VFI clusters created, two clusters operate with one V/F value while the other two operate with another V/F value. Thus, half the cores are assigned to operate with a frequency of \( f_1 \) while the remaining cores are assigned to operate with a frequency of \( f_2 \) (with \( f_2 < f_1 \)). While executing Word Count on the 64-core system with a large input dataset (100 MB input), the default phoenix scheduler creates 100 map tasks at the beginning of the Map phase and each core starts to perform Map operation on one of the 100 tasks. For each core, the duration spent on this initial Map task depends on the data unit assigned to it and the frequency of operation. The execution duration range (in seconds) for the initial Map task is given by:

- Cores with frequency \( f_1 \): 0.268 to 0.284 (average: 0.270)
- Cores with frequency \( f_2 \): 0.280 to 0.324 (average: 0.320)

Thus, certain cores with the low frequency \( (f_2) \) may finish earlier than the cores with the high frequency \( (f_1) \), depending on the data assigned. With the default task stealing mechanism, the low frequency cores that finish their initial Map task before 0.284 seconds would steal one of the unfinished tasks from cores running with \( f_1 \), leaving specific high frequency cores with no task to perform after 0.284 seconds. However, more often than not, such a stealing would result in an execution time penalty, as the average task execution duration on cores with frequency \( f_2 \) is higher than that on cores with frequency \( f_1 \). Hence for VFI systems, it is necessary to modify the default task stealing mechanism in such a manner that the cores with higher V/F do not remain unnecessarily idle. To prevent the cores with lower V/F from performing an
undesired task stealing, we restrict the number to tasks performed by cores with lower V/F to $N_f$ defined by (3).

$$N_f = \frac{N}{C} \times \left(1 - \frac{f_{\text{max}} - f}{f_{\text{max}}} \right), \quad \forall f < f_{\text{max}}$$

where $N$ is the total number of tasks to be performed, $C$ is the number of cores, $f$ is the frequency assigned to the core and $f_{\text{max}}$ is the maximum frequency of operation. Fig. 3 presents the complete VFI platform design flow. The system with initial clustering is denoted as VFI 1. The realignment of V/F values for clusters in VFI 1 results in the final VFI system, named VFI 2 system.

5. Wireless NoC Architecture

A WiNoC architecture, where the long-range shortcuts are implemented through mm-wave wireless links operating in the 10-100 GHz range, is shown to improve the energy dissipation profile and latency characteristics of multicore chips. This WiNoC can be used as the communication backbone for a VFI-partitioned multicore platform. In this work, we design the WiNoC architecture to support efficient data exchanges among various VFI domains running MapReduce. This is done by creating the wireline network, mapping threads and placing the wireless links using the knowledge of the VFI domains and their traffic characteristics.

In the WiNoC, the wireline links are designed using a power-law model [19]. We assume an average number of connections, $(k)$, from each NoC switch to the other switches. The value of $(k)$ is chosen to be four so that the WiNoC does not introduce any additional switch overhead with respect to a conventional mesh. Also, an upper bound, $k_{\text{max}}$, is imposed on the number of ports attached to a particular switch so that no switch becomes unrealistically large. This also reduces the skew in the distribution of links among the switches. There is no specific lower bound on the number of ports attached to a switch but a fully connected network implies that this number must be at least 1. Both $(k)$ and $k_{\text{max}}$ do not include the local NoC switch port to the core.

Due to the nature of the VFI clustering, additional constraints need to be applied to the connectivity of the WiNoC. The distribution of links is divided into two steps: VFI intra-cluster connections need to ensure each cluster’s connectivity, and VFI inter-cluster connections, to enable communication between the clusters. This is to ensure that both intra-cluster and inter-cluster communications have sufficient resources and none of them becomes a bottleneck in the overall data exchange.

For each switch, $(k)$ is divided into two parts, $(k_{\text{intra}})$ and $(k_{\text{inter}})$, the average number of intra-cluster and inter-cluster connections to other switches respectively. For the VFI intra-cluster connections, each cluster is treated separately. A network is created for each cluster such that the connectivity follows the power-law model; the network cluster is fully connected and has an average intra-cluster connectivity, $(k_{\text{intra}})$.

The VFI inter-cluster connections are created such that the connectivity also follows the same power-law model as the intra-cluster connections, has an average inter-cluster connectivity, $(k_{\text{inter}})$. The number of links going from one cluster to another is decided by the inter-VFI traffic. The proportion of links allocated between two clusters is directly related to the proportion of inter-cluster traffic between the two clusters in total inter-cluster traffic.

6. Wireless Link Placement and Thread Mapping

To help facilitate a predominantly inter-cluster communication, we use mm-wave wireless links to interconnect switches that are separated by long distances. In [8], it is demonstrated that it is possible to create 3 non-overlapping channels with on-chip mm-wave wireless links. 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. Each of these wireless ports will have a wireless interface (WI) tuned to one of the three wireless channels. The optimum number of WIs is 12 for a 64-core System size [20]. Therefore, we assign three WIs, working in the three non-overlapping channels, to each of the 4 clusters.

In this work, we propose two methodologies to place the wireless links and to map threads; one such strategy is to minimize the traffic-weighted hop count while the other one is to maximize the wireless utilization. The first methodology maps the threads in their specific VFI configurations in order to minimize the distance of highly communicating cores. Then, the WiNoC network is created as described above. Simulated annealing is finally used to find the optimal WI placements that minimize the average traffic-weighted hop-count.

In the second methodology, we aim at increasing the amount of traffic going through the wireless links. The wireless nodes are placed near the center of each VFI cluster. This allows most of the cores to have wireless access opportunities. Then the thread mapping uses the idea of logically near, physically far (by using the wireless links) to place highly communicating threads closer to the WIs. This strategy enables the use of energy-efficient wireless links for a higher number of flits without compromising performance. We will choose between the minimized hop-count and maximized wireless utilization wireless placement methodologies depending on their achievable performances.

7. Experimental Results and Analysis

In this section we evaluate the performance of a 64 core multiprocessor system running Phoenix++, in the presence of the above discussed VFI clustering and WiNoC interconnection architecture. We use GEMS [21], a full system simulator, to obtain detailed processor and network-level information. We consider a system of x86 cores running Linux within the GEMS platform for all experiments. The memory system is MOESI_CMP_directory setup with private 64KB L1 instruction and data caches and a shared 32MB (512KB distributed per core) L2 cache. The width of all wired 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. The NoC simulator uses switches synthesized from an RTL level design using TSMC 65-nm CMOS process, using Synopsys™ Design Vision. All ports except those associated with the 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 the token [8].

Energy dissipation of the network switches, inclusive of the rerouting block, 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 [22]. In this work we evaluate our methods using six Phoenix++ applications, namely Histogram (HIST), Kmeans, Linear Regression (LR), Matrix Multiplication (MM), PCA, Word Count (WC). Among the Phoenix++ applications analyzed, Kmeans and PCA have two MapReduce iterations, whereas all other applications have a single MapReduce iteration.

<table>
<thead>
<tr>
<th>Application</th>
<th>Input dataset size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiplication</td>
<td>Matrix with dimension 999 x 999</td>
</tr>
<tr>
<td>Kmeans</td>
<td>Vectors with dimension of 512</td>
</tr>
<tr>
<td>PCA</td>
<td>Matrix with dimension 960 x 960</td>
</tr>
<tr>
<td>Histogram</td>
<td>Medium (399 MB)</td>
</tr>
<tr>
<td>Word Count</td>
<td>Large (100 MB)</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Medium (100 MB)</td>
</tr>
</tbody>
</table>
iteration. Further information on the applications analyzed is provided in Table 1.

### 7.1 Impact of Bottleneck cores

First, we determine the V/F pairs for all clusters in VFI 1 configuration. Then, as we have discussed in section 4.2, certain Phoenix++ applications (PCA, HIST, MM) indicate the presence of bottleneck cores and require a reassignment of initial V/F values. In this section, we analyze the impact of bottleneck cores on the execution time of these applications. Fig. 4 presents the execution times and Energy Delay Products (EDP) of PCA, HIST and MM for the two VFI systems, the initial VFI system (VFI 1 in Fig. 3) and final VFI system with V/F reassignment (VFI 2 in Fig. 3). The shown execution time and EDP values are normalized with respect to a baseline non-VFI system with mesh interconnection (NVFI Mesh). For all these three applications, the V/F value of the cluster with bottleneck cores was increased to 1.0V/2.5 GHz in VFI 2 system from 0.9V/2.25 GHz in VFI 1 system. The V/F values of all other clusters remain unchanged. Table 2 shows the V/F values for all the benchmarks in both the VFI configurations (reassigned V/F values are highlighted).

From Fig. 4 (a), we can observe that PCA benefits most by reassigning the V/F values, followed by MM. The reason for the execution time improvement in VFI 2 system can be understood from Fig. 5. Fig. 5 presents the average and bottleneck cores’ utilization values for PCA, HIST and MM. PCA has the highest bottleneck core to average core busy utilization ratio, followed by MM. Higher this ratio, higher is the impact of bottleneck core on the execution time. As explained in section 4.2, with the initial V/F assignment, bottleneck cores were placed in clusters with low V/F values. This slows down the bottleneck cores, leading to VFI 1 system experiencing a higher execution time penalty when compared to the NVFI mesh. However, by increasing the V/F values of the whole cluster we may lose energy saving opportunity. We can observe from Fig. 4 (b) that this execution time improvement in VFI 2 system has been achieved without noticeable increase in the energy consumption. EDPs of the PCA and MM are improved for the VFI 2 system compared to the VFI 1 system whereas HIST pays no EDP penalty. Hence, we can conclude that the improvement in execution time due to V/F re-assignment is more than the decrease in energy savings.

### 7.2 Network Performance

In this section we evaluate the optimal network parameters for the WiNoC to be used with the MapReduce applications under consideration. To do so we focus on the network parameters, wireless methodology, \((k_{\text{intra}})\) and \((k_{\text{inter}})\) parameters discussed in section 5. We vary each of these parameters and use their network energy-delay-product (EDP) as the performance metric. Due to the nature of fully connected networks, our VFI clusters of 16 cores require a \((k_{\text{intra}})\) of at least 1.875. As the value of \((k)\) is 4, the values of \((\langle k_{\text{intra}}, k_{\text{inter}}\rangle)\) can be either (3,1) or (2,2). We have found that (3,1) configuration always performs better than (2,2). Hence this network configuration is used in all further performance analysis. Fig. 6 demonstrates the reduction in EDP when using the maximized wireless utilization over the minimized hop-count strategy. For all benchmarks considered, the system with maximized wireless utilization and \((\langle k_{\text{intra}}, k_{\text{inter}}\rangle)\) values equal to (3, 1) perform consistently better than other alternatives.

### 7.3 Full-System Performance

For VFI-based designs, one of the main goals is to reduce energy while maintaining an acceptable level of performance degradation. So, in order to evaluate our VFI-enabled system, we present both execution time and the whole system EDP parameters.

First we consider the overall execution time of the designed system with respect to the baseline non-VFI (NVFI) mesh-based system. Fig. 7 shows the execution time relative to NVFI mesh for both VFI mesh and VFI WiNoC. As expected in the presence of VFI, traditional mesh-based designs suffer a degradation of up to 10.5% of execution time due to the lowered V/F values. Since the WiNoC is able to transfer data across the network faster giving rise to faster non-local L2 access, the performance degradation is lowered and eliminated for certain benchmarks under consideration. MM, WC, LR, and Kmeans were all able to execute quicker using WiNoC-based VFI than NVFI mesh.

Among all the applications, WC and Kmeans achieve the highest execution time improvements with an improved interconnection network (15% and 14% respectively). Both applications have a high number of MapReduce \(k\text{eys}\) and also exhibit high amount of traffic interaction between the cores that are distant. With an improved

---

**Table 2. V/F assignments for MapReduce Applications**

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Cluster</th>
<th>Cluster</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (V/GHz)</td>
<td>2 (V/GHz)</td>
<td>3 (V/GHz)</td>
<td>4 (V/GHz)</td>
</tr>
<tr>
<td>MM</td>
<td>1.0/2.5</td>
<td>0.9/2.25</td>
<td>1.0/2.5</td>
</tr>
<tr>
<td>HIST</td>
<td>1.0/2.5</td>
<td>0.9/2.25</td>
<td>1.0/2.5</td>
</tr>
<tr>
<td>Kmeans</td>
<td>0.6/1.5</td>
<td>0.6/1.5</td>
<td>0.8/2.0</td>
</tr>
<tr>
<td>WC</td>
<td>0.8/2.0</td>
<td>0.8/2.0</td>
<td>1.0/2.5</td>
</tr>
<tr>
<td>PCA</td>
<td>0.9/2.25</td>
<td>0.9/2.25</td>
<td>0.9/2.25</td>
</tr>
<tr>
<td>LR</td>
<td>1.0/2.5</td>
<td>1.0/2.5</td>
<td>0.9/2.25</td>
</tr>
</tbody>
</table>

---

**Figure 4: Comparison between VFI 1 and VFI 2 systems**

**Figure 5: Core utilization Values**

**Figure 6: EDP of the maximized wireless usage relative to the minimized hop count wireless placement methodology**

**Figure 7: Normalized execution time of each Phoenix++ execution operation with respect to non-NVFI mesh.**
network, the latencies in transferring the Keys and Values among the cores get highly reduced, leading to better execution times. Among all the applications studied, LR application has the highest traffic due to its near-core heavy communication pattern and dependency on core-main memory interaction, the improvement achieved for LR with WiNoC is limited (4%). Due to the trade-offs associated with reducing energy at the cost of performance, the energy-delay-product (EDP) is the most relevant metric when analyzing the energy profile. By delay we consider the execution time here. Fig. 8 shows the EDP for the VFI configurations in relation to NVFI mesh. The variation of the differences between the VFI Mesh and VFI WiNoC EDP arise from each application’s core interaction rates. For example, LR has the greatest core interaction rate and difference between the VFI mesh and VFI WiNoC EDP. This can be attributed to WiNoC having lower energy consumption compared to the Mesh architecture.

Furthermore, it can be observed from Fig. 8 that Kmeans achieve the highest EDP savings with respect to NVFI mesh. As explained in section 4.2, the core utilization values for Kmeans varies highly over the 64 cores. It can be seen in Fig. 2, that about 32 cores have less than 50% utilization when compared to the average utilization. Such cores can be operated with significantly a lower V/F, without affecting the execution time leading to higher energy savings.

In summary, we can see that for all of the benchmarks considered, we are able to save significant EDP by using WiNoC over NVFI mesh, (maximum 66.2% and 33.7% on average). These large reductions in EDP are due to enhancement of the energy dissipation profiles of both the processing cores and the interconnection network without noticeable compromise in execution time. This demonstrates the ability for WiNoCs to enable VFI-based designs, greatly reducing energy without impacting the performance of traditional mesh-based systems.

8. Conclusion

In the context of meeting the growing demand for big data analytics, designing a low power and high performance multicore-based MapReduce platform is essential. In this paper, we have presented an energy efficient multicore architecture, which integrates VFI and emerging wireless interconnects in a single platform, targeting MapReduce applications. In an effort to achieve energy efficiency on both processing cores and interconnect fabrics, we have integrated VFI and emerging wireless interconnects in a single platform. We have illustrated that with efficient V/F assignments, it is possible to achieve significant energy reduction for MapReduce applications while keeping the execution time penalties to a minimum. The use of wireless interconnects leads to energy and execution time improvements. The proposed WiNoC-enabled VFI design achieves on an average 33.7% EDP improvement over a baseline non-VFI platform with mesh NoC.

9. Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) grants CCF-0845504, CNS-1059289, CNS-1128624, and CCF-1162202, GRC 2012-TJ-2232, as well as Army Research Office grant W911NF-12-1-0373.

10. References