VERVE: A Framework for Variation-Aware Energy Efficient Synthesis of NoC-based MPSoCs with Voltage islands

Nishit Kapadia, Sudeep Pasricha
Department of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, U.S.A.
nkapadia@colostate.edu, sudeep@colostate.edu

Abstract

With feature sizes far below the wavelength of light, variations in fabrication processes are becoming more common and can lead to unpredictable behavior in modern multiprocessor system-on-chip (MPSoC) designs. The design costs associated with margining required to overcome this unpredictability can be prohibitively high. System-level design approaches that are aware of these variations can be crucial for designing energy-efficient systems. We note that by performing voltage island placement appropriately, the two major unintended consequences of variations on the circuit characteristics (altered delay and power dissipation) can be traded-off, in order to minimize overall system energy. To this end, we propose a novel design-time system-level synthesis framework that is cognizant of process variations while mapping cores operating at specific supply voltages to a die and allocating communication routes on a 2D-mesh network-on-chip (NoC) topology for optimal energy-efficiency. Our experiments with real-world and synthetic application benchmarks show that our framework achieves 3.4% savings in computation energy and 19% savings in communication energy compared to the best known prior work on NoC-based MPSoC synthesis that considers process variations.

Keywords

System-level CAD, NoC synthesis, core mapping, computation energy, multi-core systems, process variations

1. Introduction

With CMOS feature sizes shrinking far below the wavelength of light, variations in fabrication processes are becoming more and more prevalent today, causing unpredictable behavior in modern multiprocessor system-on-chip designs (MPSoCs). The trends of increasing number of critical paths and decreasing logic depth in MPSoCs contributes to the growing impact of within-die (WID) variations on these designs [1]. WID variations are often classified into *systematic* and *random* variations. Systematic variations are caused by aberrations in the lithographic lens, whereas random variations are a result of unequal densities of dopant atoms. While random WID variations are present at a much smaller (transistor-level) granularity, systematic variations exhibit strong correlation over larger areas. In modern MPSoCs where individual cores are small enough, spatially correlated systematic variations manifest across multiple cores, leading to core-to-core (C2C) variations [2].

One of the main effects of process variations is the deviation of the threshold voltage (V_T) from its nominal value. A rise in the value of parameter V_T increases circuit delay and at the same time reduces leakage power. Conversely, a reduction in V_T decreases circuit delay and increases leakage power. In the design of modern server and desktop systems, performance per watt (energy-efficiency) is a key metric to be optimized. The total energy consumption required for executing a certain application task also determines the feasibility of porting it onto a battery-operated mobile device. Variations in V_T in all of these systems can cause violations of power/energy budgets which can make cooling costs prohibitive or even cause system-wide failure. Therefore, for systems to operate within their energy budgets without compromising on performance, *system-level methodologies* for variation-aware energy-efficient MPSoC design are sorely needed.

As the number of cores integrated in MPSoCs continues to increase into the hundreds [3], [4], the complexity of network-on-chip (NoC) fabrics [5] required to interconnect communicating cores on a chip has also increased. NoCs have been shown to dissipate significant power (e.g., ~30% of chip power in Intel's 80-core teraflop [3] and ~40% of chip power in MIT RAW [4] chips). As a result, reducing not only computation but also communication power has become a high priority for chip designers. The use of voltage islands (*VIs*) can limit the overhead of implementing power management schemes by combining cores into islands that use the same V_{DD} and ground lines. Compared to managing power separately for each core, *VI*s allow minimizing the number of VLCs (voltage level converters) and MCFIFO (multiple clock first in-first-out) queue based frequency level converters required [6]. However, *VI*s complicate the problem of NoC design, requiring designers to revisit the problems of *VI* partitioning, core-to-tile mapping, and routing path allocation so that computation and communication power can be minimized.

It is well known that a rise in the value of V_T (due to process variations) on any tile of the mesh increases circuit delay and decreases the leakage power of the corresponding core; on the other hand, a reduction in V_T decreases circuit delay and increases leakage power. As the basis of this work, we note that by performing *VI* placement appropriately, the two major unintended consequences of V_T variations on the circuit characteristics (delay and power dissipation) can be traded-off, to minimize system energy. To this end, we propose a novel design-time system-level synthesis framework that is cognizant of process variations. Our framework (*VERVE*) takes advantage of variation-maps for a given process [14] while mapping cores operating at specific supply voltages to a die and allocating communication routes on a regular 2D-mesh NoC topology for optimal energy efficiency. Our experiments on several real-world and synthetic application benchmarks, and across different NoC mesh sizes show that *VERVE* achieves notable energy improvements over the best known prior work on variation-aware synthesis of NoC-based MPSoCs [7].

2. Related Work

The problem of variation-aware system-level design of MPSoCs has been introduced only in recent years and has not been studied extensively. Hong et al. [9] propose a variation-aware thread mapping scheme to mitigate the effects of non-uniform frequencies over a multi-core system. Garg et al. in [10] show that systems implemented with multiple *VIs* isolate the effects of WID process variations to a certain frequency domain and thus are likelier to meet throughput constraints compared to fully synchronous counterparts. Herbert et al. [11] improve energy efficiency by modifying existing DVFS control algorithms to shift work from inefficient, leaky processing units to efficient and less leaky ones, thus exploiting process variability. Wang et al. [12] model task completion time as a stochastic variable and generate both task scheduling and routing procedures to optimize the probability of a given schedule meeting performance constraints. The authors in [14] propose variation-aware scheduling algorithms to save power and improve throughput, while using the knowledge of variation-distribution on the CMP die. Given the probabilities of operating frequencies of hardware resources on an MPSoC, Mirzoyan et al. [13] show significant improvements in timing yield with their WID-variation aware task mapping approach for real-time streaming applications. Mazjoub et al. [7] propose a *VI*- and core- placement approach to balance between limits of spatial extent of the WID variations across cores, and the communication patterns between *VI*s by varying the shape of *VI*s in order to minimize total power. In contrast to these approaches, our VERVE framework optimizes energy dissipation by trading-off the effects of process variations. One of our main contributions is a novel *VI-placement* approach which is aware of WID process variations and optimizes energy efficiency. *To the best of the authors' knowledge, this is the first work which addresses the problem of determining VI locations on MPSoCs to mitigate effects of process variations.*

The problem of NoC synthesis with multiple supply *VI*s on the other hand, has been addressed in several works. For example, Ogras et al. [15] perform *VI* partitioning and static voltage-frequency assignment on a pre-mapped NoC to optimize communication and energy consumption while meeting task deadline constraints. Jang et al. [16] showed improvements over [15] to achieve less power overhead by reducing total number of MCFIFOs and VLCs needed for inter-island communication. Heuristics implementing incremental mapping on NoCs with *VIs* are used to map the cores to the die in order of their communication volumes. In this paper, we build on the concept of incremental swaps from prior work, and present a *novel core mapping technique that uses a more efficient distributed decision making process, as opposed to a central one used in prior work.*

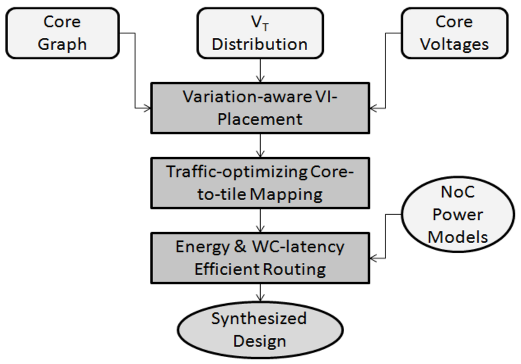
Core Graph
V_T Distribution
Core Voltages
Variation-aware VI-Placement
Traffic-optimizing Core-to-tile Mapping
NoC Power Models
Energy & WC-latency Efficient Routing
Synthesized Design

Figure 1: Overview of the VERVE framework design flow

3. Problem Formulation

We choose to focus on the problem of synthesizing a regular NoC topology because it is widely expected that irregular topologies will be impractical to design and ensure predictable transfers for in large MPSoC systems with hundred or more cores. Moreover, recent industrial multiprocessor designs (e.g., Intel's Teraflop and SCC processors, and Tilera's Tile64) all make use of regular topologies. The inputs to our problem are as follows:

- A regular mesh-based 2D NoC with T tiles: $T = (d^2)$, where d is the dimension of the mesh, and each tile consists of a compute core, a NoC router, and a network interface (*NI*) component between them;
- A process-variation map (V_T-map) obtained from the process variation model, that provides the T threshold voltage (V_T) values assigned to T tiles of the mesh;
- A core graph $G(V, E)$ with the set of N vertices such that T=N, with vertices $\{v_1, v_2, v_3,\ldots,v_N\}$ representing homogeneous cores on which tasks have already been mapped; and the set of M edges $\{e_1,e_2,e_3,\ldots,e_M\}$, where edge weights $\{w(e_1), w(e_2),\ldots,w(e_M)\}$ represent communication dependencies (communication volumes) between cores;
- A set of Ω supply voltage (V_{dd}) levels, which also represents the total number of *VI*s;
- Pre-assigned supply voltage level for each core $\{v_{dd}(V_1), v_{dd}(V_2), \ldots., v_{dd}(V_N)\}$ to meet compute performance requirements of tasks mapped to the cores, corresponding to the highest V_T value;
- Baseline processing times for each core $\{t(V_1),t(V_2), \ldots.,t(V_N)\}$ based on the compute requirements of task(s) mapped to cores. The times are calculated for the maximum allowed operating frequency (*max_freq*) at the highest supply voltage level and nominal value of threshold voltage.

Objective: Given the above inputs, the goal of our work is to obtain a core-to-tile mapping and synthesize a regular 2D mesh NoC architecture for a specific application, such that all cores within individual voltage islands are contiguously placed, while minimizing total energy consumption in compute cores and communication resources (including network routers, data links, VLCs, and MCFIFOs).

4. VERVE Framework Overview

In this section, we present an overview of our variation-aware and energy-efficient VERVE framework for synthesizing NoC-based MPSoCs. Figure 1 shows the high level flow of our synthesis framework. The problems addressed in the three major steps of our framework are summarized below: *(i) Variation-aware VI-placement:* Given the threshold voltage variation distribution over the die and the number of cores in each *VI* (which depends on the application-specific minimum compute performance requirements), this step assigns operating voltages to individual tiles considering the suitability of mapping cores operating at specific supply voltages to certain tiles for optimal energy efficiency; *(ii) Traffic optimizing core-to-tile mapping:* Given the locations of Ω contiguous *VI*s on the mesh, this step maps cores to tiles (within their respective voltage islands) to minimize total traffic in the NoC; and *(iii) Energy and worst case (WC) latency efficient minimal routing:* This step allocates paths for communication flows such that the number of VLCs and MCFIFOs used are minimized for inter-island data communication, while also minimizing worst-case latency. The following sections elaborate on each of these three steps.

4.1 Variation-aware VI placement (*VI*-to-tile mapping)

As discussed earlier, a rise in the value of V_T (due to process variations) on any tile of the mesh increases circuit delay and decreases the leakage power of the corresponding core; on the other hand, a reduction in V_T decreases circuit delay and increases leakage power. Increased circuit delay, reduces the maximum frequency that a processing core can be clocked at. Moreover, as operating frequency scales with supply voltage, the operating frequency and power dissipation of a core (and thus its energy) vary with different values of V_{dd} as well as V_T. In order to enable a trade-off between delay and power, a methodology for voltage assignments of the tiles that is aware of the V_T variation map is essential. Thus, given the number of cores within each *VI* and the V_T values corresponding to tiles on the mesh, our *VI-placement* algorithm performs voltage assignments to the T tiles such that the *VI*s are contiguous and compute energy for the given application is minimized. Here, we approximate the variations of the circuit parameter V_T as discrete values (in fixed increments/ decrements) relative to the nominal V_T value. Let Ψ be the total number of possible V_T values. Initially, all T tiles (with pre-defined V_T values) are placed in Ψ V_T*-baskets* and the Ω *VI*s are empty. The goal of the *VI-placement* algorithm is to transfer all the tiles from the V_T*-baskets* to the *VI*s (as shown in Figure 3) to produce an energy-efficient and contiguous set of *VI*s mapped on the 2D mesh. Note that multiple V_T*-baskets* would possibly prefer the same *VI* for minimum energy; conversely, multiple *VIs* can be affine to the same V_T*-basket*. Therefore, our *VI-placement* algorithm systematically prioritizes the *VI*-to-tile mapping in order to maximize energy savings. The three major phases of this algorithm are discussed below.

4.1.1 *VI* Preference List Construction

A compute energy table of dimensions $\Omega\times\Psi$ (with Ω rows and Ψ columns; Figure 2) is first created using an average core processing time (defined as the sum of processing times of all tasks mapped to that core) for the given application core-graph:

$$E_{compute} = \left(P_{static} + P_{dynamic}\right) * D$$

The average core processing time (D) for each entry in the energy table, is obtained by scaling the average baseline processing time with respect to the operating frequency for the corresponding $\{V_T,\ V_{dd}\}$ pair. Corresponding to each entry in the table a relative energy (*rE*) is computed which represents the energy dissipation for the particular $\{V_T,\ V_{dd}\}$ pair with respect to all possible V_T values (w.r.t. the energies of the entire j^{th} row as shown in Figure 2), using the following equation:

$$rE(k,j) = {E(k,j)}\Big/{\textstyle\sum_{x=1}^{x=\psi} E(x,j)}$$

Each V_T*-basket* here is associated with a *VI-preference-list* constructed using its column in the relative-energy-table (k^{th} column as shown in Figure 2) sorted in increasing order of relative energies, such that the V_{dd} entry with the lowest relative energy will have the highest preference on the *VI-preference-list*. Energy dissipation is higher at higher supply voltages in general, therefore if absolute energy values (from the energy table in Figure 2) are used to build the *VI-preference-lists*, lower voltages will almost always get higher preferences (priorities) in the *VI-preference-list.* On the other hand, rE_{kj} quantifies the (inverse of) suitability of mapping the j^{th} supply voltage to a tile with the k^{th} V_T value, relative to all possible (Ψ) V_T values. Thus, we build the *VI-preference-lists* (for V_T-baskets) using relative energies from the relative-energy-table shown in Figure 2.

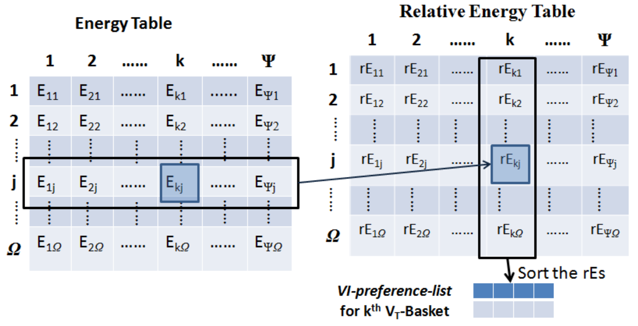

Figure 2: Constructing *VI-preference-lists* from energy table

After building the *VI-preference-lists*, the two remaining phases have the following goals: *(i) Initial Requesting* (section 4.1.2) performs preliminary mapping of contiguous *VI*s onto the tiles of the 2D mesh utilizing the energy based preference-lists of *VI*s at each $\mathrm{V_T}$-basket; *(ii) Iterative VI-Expansion* (section 4.1.3) expands *VI*s from the previous phase, again utilizing the energy-based preference-lists, while ensuring *VI* contiguity.

4.1.2 Initial Requesting

Once the V_T-baskets are set-up with the *VI-preference-lists*, the initial requesting phase involves sending tile-requests from V_T-baskets to *VI*s (Figure 3). In the beginning, all list-pointers are initialized to the highest (left-most) preferences in the *VI-preference-lists*. Then, the V_T-basket with the lowest *rE* (for the *VI-preference-list* entry currently being pointed to by the list-pointer) is selected iteratively and tile-requests are sent to the corresponding *VI* until either the *VI* is full or the *V_T-basket* is empty. In the process, if the *VI* that is currently being sent requests to becomes full, the list-pointer is advanced by one position (note that as cores in the core-graph are pre-assigned supply voltages, the capacity of each *VI* is pre-set). This process continues until all *Ω VIs* are full and *Ψ* V_T-baskets are empty (Figure 4(a)-(b)). To iteratively select the next tile, at most *Ψ VI-preference-lists* are checked for lowest *rE*. Therefore, the time-complexity of the Initial Requesting phase becomes O(*Ψ.T*), which is linear-time when *Ψ* is considered to be a constant.

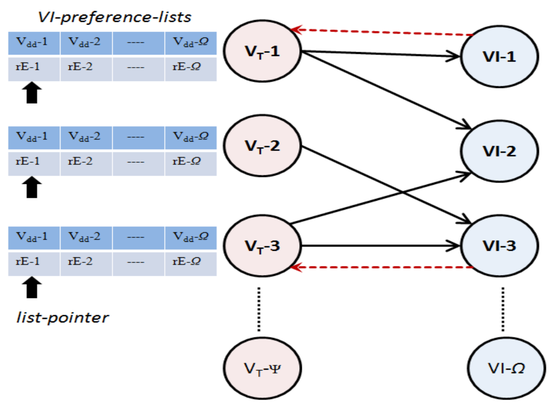

Figure 3: Initial Requesting in *VI*-placement: with *Ψ* *V_T*-baskets on the left and *Ω VI*s on the right

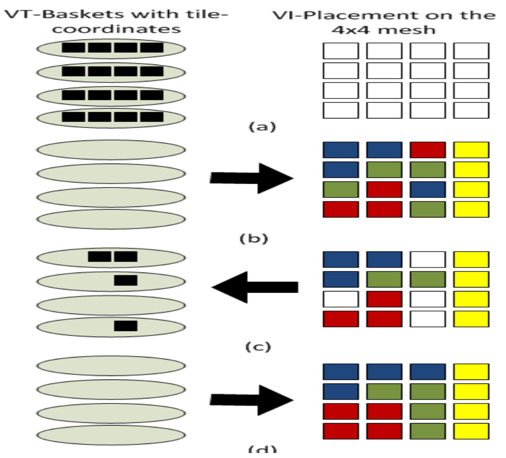
VT-Baskets with tile-coordinates
VI-Placement on the 4x4 mesh
(a)
(b)
(c)
(d)

Figure 4: Example of Initial Requesting and Iterative *VI* Expansion phases for a 4x4 mesh with *Ω*=4, Ψ=4 (each VI has 4 cores, each V_T-basket has 4 tiles); *VI*s are color-coded.

Out of all the tile-requests received by any *VI* (each tile-request defines the coordinates of its location on the mesh), the ones which belong to the biggest contiguous island of tiles are accepted (i.e., fixed on the mesh) and the rest of the tile-requests are returned to the respective *V_T-baskets* (Figure 4(c)). As an illustration, in Figure 3, as no requests are returned by *VI-2* at the end of the initial requesting phase (i.e., there are no broken arrows in red going back from *VI-2*), it is full which indicates that all tile-requests received by it together form a single contiguous island of tiles. Also, note that the *V_T-basket V_T-2* has no requests returned to it from *VI-3*, therefore, it stays empty. Both *VI-2* and *V_T-2* will no longer participate in further processing. All tile-requests accepted by *VI*s are fixed on the mesh, which implies that the tiles are assigned to a specific *VI*. On the other hand, *VI-1* and *VI-3* end up incomplete after the Initial Requesting phase as one or more tiles are returned (shown with broken arrows in red) to the respective *V_T-baskets*.

4.1.3 Iterative VI Expansion

Even though a set of contiguous tiles are allocated to each *VI* at the end of the *Initial Requesting* phase, some tiles may not have been allocated to a *VI* (e.g., Figure 4(c)). Therefore, *VI Expansion* phase incrementally assigns these remaining tiles to existing *VI*s in an energy-efficient manner, while keeping the contiguity of *VI*s intact. This step essentially scans the *VI-preference-lists* of all non-empty *V_T-baskets*, and iteratively selects a *V_T-basket* with the least relative energy (*rE*) and transfers the tiles to the corresponding non-full *VI* (selected from the *VI-preference-list*). Note that only the tiles which are adjacent to the existing *VI*s can be transferred. The pseudo-code for this phase is given below:

Algorithm 1. Iterative VI Expansion

```
input: Ψ V_T-baskets and VI-preference-lists, Ω VIs, i = 1 initially
1:  while ((∃ non-full VIs) and ($i \le$ size of VI-preference-lists))
2:    select all non-empty V_T-baskets in increasing order of their rE(i):
      {let k^th V_T-basket be selected, $1 \le k \le \Psi$ }
3:      select VI in $V_{dd}(i)[k]$: {let m^th VI be selected, $1 \le m \le \Omega$ }
4:      check for adjacency for maximum number of tiles
        from k^th V_T-basket to m^th VI
5:      transfer these tiles (obtained in step 4) sequentially to m^th
        VI, while doing a VI-status-check after each transfer
6:      if (1 or more tiles transferred in step 5) and (m^th VI is !full)
7:        Check for adjacency of tiles in all other V_T-baskets with
          $V_{dd}(j)[z]=V_{dd}(i)[k]$ ∀ j<i (where ∀ z: 1 to Ψ / z≠k)
8:        transfer these tiles to m^th VI sequentially, each
          time doing the VI-status-check
9:      end if
10:   end select
11:   increment i
11: end while
output : 2D mesh of tiles with VI-placement {M}
```

Initially, *V_T-baskets* hold the unassigned tiles and the *VI*s hold the tiles which are assigned a *VI*. The *VI-preference-lists* are scanned from highest to lowest preference (variable *i* in pseudo-code). *V_T-baskets* are selected in increasing order of relative energies, *rE(i)* entries in *VI-preference-lists* (**line 2**). *$V_{dd}(i)[k]$* (which represents the *V_{dd}* value of the *i*th entry in the *preference-list* of *k*th V_T*-basket*) is selected as the target *VI* for each new *V_T-basket* (**line 3**). The *k*th *V_T-basket* is checked for maximum possible number of tiles that can be adjacent to the target *VI* (**line 4**). These tiles are transferred to the target *VI*. After every single transfer, a *VI-status-check* needs to be performed (**line 5**), which checks if the *VI* is full, and if so, nullifies entries of this *VI* from all *VI-preference-lists* and discontinues tile transfers to it. Now, with the augmented form of the target *VI* (iff at least one tile is transferred since the start of the current *i*th iteration), tiles in other *V_T*-baskets that were not adjacent to the target *VI* during earlier iterations (higher on the *VI-preference-lists*: *j<i*), can now potentially be part of it. Therefore, tiles in all *V_T-baskets* with the target *VI*, i.e., *$V_{dd}(i)[k]$*, on their *VI-preference-lists* at an earlier index (*j<i*) are checked for adjacency and transferred (**lines 6-8**). This process continues until either all *V_T*-baskets become

empty (all *VI*s are full) or the *VI-preference-lists* have been scanned completely (loop-condition in **line 1**). Finally, a set $\{M\}$ representing a 2D-mesh of voltage assigned tiles is produced as shown in Figure 4(d). In summary, the *VI-Expansion* phase scans up to Ω *VI-preference-list* entries of Ψ V_T*-baskets* (up to $\Omega.\Psi$ iterations of select (steps 2-10)). For each of these iterations at most (Ψ-1) other *VI-preference-list* entries are scanned, where all remaining tiles in the current V_T*-basket* (less than T tiles) are evaluated for adjacency to the *VI* under consideration. Therefore, the time complexity of the *VI-Expansion* phase becomes O($\Psi^2.\Omega.T$), which is also linear-time as Ψ and Ω are constants.

Note that for some input instances, even after performing the iterative VI expansion, a few tiles may still remain unassigned in the V_T*-baskets* (i.e. one or more *VIs* may remain non-full). This can happen when all peripheral tiles of certain *VI* are occupied by neighboring *VIs* thus making its expansion impossible. To resolve such a situation and form Ω *VIs* on the die with all T tiles, we employ a simple post-processing step. The post-processing step basically finds a path of *VIs* from the unassigned tile to the non-full island, and executes a sequence of tile transfers (between islands) along this path of *VIs* to occupy the unassigned tile.

4.2 Traffic Optimizing Core-To-Tile Mapping

After the *VI*-to-tile mapping step, the objective of this second step is to minimize inter-core NoC traffic. Total traffic is the sum of products of Manhattan distances and communication volumes of all individual communication flows. Mathematically, this can be expressed as: $\sum_j MD_j * w(e_j)$; where j is a uni-directional communication flow between any two cores on the die. The contribution of an individual core to the total traffic is defined as *core-traffic*, which includes both ingress (i.e., incoming) and outgress (i.e., outgoing) traffic. Note that *net core-traffic* of the i^{th} core in x-direction is the absolute difference between *core-traffic*$_i\{+x\}$ and *core-traffic*$_i\{-x\}$. The net core-traffic in the y-direction is defined similarly.

Algorithm 2. Core-To-Tile Mapping

```
input: core graph G (V, E), VI-placement {M}
1: generate initial core-to-tile mapping, retaining {M}
2: while (d consecutive aborts are not encountered)
3:     choose the core with maximum total net core-traffic
4:     choose the direction of most net core-traffic
5:     check the validity of the swap
6:     if the swap is invalid, consider other directions
7:     if no directions valid, abort; restrict the core from swapping
       for the next d iterations
8:     if a valid swap exists, perform the swap
9: end while
output : traffic-optimized final mapping
```

Algorithm 2 above describes the key steps in our core-to-tile mapping, which eventually decreases the total traffic in the mesh NoC. The cores are initially mapped in an arbitary manner to the tiles given the constraints imposed by the *VI-placement* step. After this initial mapping (**line 1**), all communicating cores have some *net core-traffic* values associated with them, in the x and y directions. Our *incremental swap algorithm* in this step attempts to iteratively decrease the Manhattan distance of the core with the highest *core-traffic* in the entire mesh by swapping it with another core (in the direction of most *core-traffic* for that core) (**lines 3, 4**). The swaps are allowed to occur between neighbors either in the x-direction, y-direction, or diagonally on the mesh structure. Swaps can of course take place only within the same voltage island so as to retain the *VI-placement* on die produced in the previous step.

To explore the mapping space systematically (in a directed way) swaps that do not reduce total traffic in the NoC, as well as swaps (in a Tabu list) that move cores in the direction that they were swapped from in the past, are invalidated (**line 5**). If a swap is valid (**line 8**), then the core-traffic distribution of the mesh is updated after the swap, and the complementary move direction for the core is added to the Tabu list of that core (e.g., -y if the core moved in the +y direction), so that the core will never move back in the direction it was swapped from. If a diagonal swap takes place, the corresponding x and y directions are added to the Tabu list. If a valid swap is not found (**lines 6, 7**), it is aborted and the corresponding core is restricted from swapping for the next d (dimension of the mesh) number of iterations. The *incremental swap algorithm* rapidly decreases the Manhattan distances of high communication-volume flows in the mesh NoC. Eventually, a state of equilibrium is reached, where valid swaps that reduce total NoC traffic are no longer readily available; (i.e. until d consecutive aborts are encountered, loop-condition in **line 2**); at which point the algorithm terminates. Note that cores are never swapped back in the direction of their previous locations; this gives us a theoretical upper bound on the total number of swaps that the N cores in the mesh can undergo: O($2N.(N^{1/2}$-1)).

4.3 Energy and WC-latency Aware Routing

The core-to-tile mapped mesh consists of multiple *VIs* that run on different voltage levels as well as different frequencies. Therefore, for inter-island communication, voltage level converters (VLCs) and frequency level converter resources (MCFIFOs) are required in the corresponding routers. Whenever a low voltage core transmits to a higher voltage core, a VLC is needed on the outgoing port of the source router. Also, for any inter-island link, an MCFIFO is needed for the higher frequency core as the connecting link works at the lower frequency. These frequency and voltage conversion components incur an overhead in terms of power dissipation and delay. Also, in a general pipelined routing architecture, the link delay on the slowest link on the path becomes the lower bound on path latency. Thus, the main objective of our routing path allocation is to *find a path for each communication flow such that the number of inter-island links and the bottleneck volume (highest communication volume allocated on any link along the path) are reduced.*

The order in which the communication flows are routed, is determined in the following way in our framework. The communication flows with longer minimal paths (MDs) have more choices for routing and thus have a larger scope for optimization. Also, flows with smaller volumes have a smaller overall transfer latency footprint for NoC communication. Therefore, communication flows are sorted in the increasing order of their path lengths, in decreasing order of their communication volumes for the same path length; and considered for routing in that order in this third step of our framework. For each communication flow, we consider all candidate minimal paths. Note that, the number of all possible minimal paths between two cores on a 2D mesh, which are K hops apart ($K = x + y$; where x and y are the number of x-hops and y-hops on the path) is given by ${}^{K}C_{x}$. Out of these candidate minimal paths, we choose a routing path based on the following optimization objectives (in that order):

1) Minimize energy overhead of MCFIFOs and VLCs; and
2) Minimize latency bottleneck on the communication path

To meet the above objectives, we first choose paths that need the minimum total number of inter-island links. Then, out of the chosen paths (with the same number of inter-island links), we choose the one which has the lowest bottleneck volume. When a path is chosen, the current communication flow (volume) is allocated to its constituent links. We also perform a post-processing design time cyclic dependency analysis, based on an approach from prior work [23], to ensure freedom from cyclic dependencies which can lead to deadlock at runtime. After the completion of this step, a voltage-assigned, mapped and routed NoC-based MPSoC is obtained for the given application.

5. Experiments

5.1 Experimental Setup

We use the ARM Cortex-A9 multi-core processors [17] as the baseline MPSoC compute cores in our experiments, which support five operating voltage levels (Ω=5): 0.8V, 0.9V, 1.0V, 1.1V and 1.2V; and a maximum nominal operating frequency (at V_{dd}=1.2V & V_T=0.4V) of 1.988 GHz. The V_T-map is generated using the open-source tool [22] (based on systematic and random WID-variation model in [8]), for different mesh sizes [(7x7), (8x8) and (10x10)]. The values of 0.4 and 0.09 are used for the statistical mean and a standard deviation of the parameter V_T respectively and a value of □=0.5 is used (as recommended in [8]). For the *VI-placement* step, seven *V_T-baskets* with discrete V_T values ranging from 0.34V and 0.46V, in increments of 0.02V (0.4V being the nominal V_T) are used; where, the continuous distribution of V_T values in the V_T-map are rounded to the nearest discrete value.

Our experiments were conducted using a mix of ten random task-graphs (*Rand_50, Rand1_100, Rand2_100, Rand3_100, Rand4_100, Rand1_300, Rand2_300*) and real-world application task-graphs (*Sparse_96, Robot_88, Fppp_334*) taken from the Standard Task Graph (STG) set [18], where each vertex (task) corresponds to processing time (in units of 100's of ms) and the edge weights represent inter-task communication volumes (with values ranging between 20 Gb and 220 Gb). In order to convert task-graphs into core-graphs, we iteratively shrank the inter-task edges with the lowest sum of processing-times of the two tasks it connects, until the number of tasks (G) equaled the number of tiles or cores (N) on the 2D-mesh (assuming G>=N), for a given mesh size. In this way, the benchmark *Rand_50* (with 50 tasks) is converted to a 49-core core-graph to be mapped on a (7x7) mesh. Similarly, *Robot_88, Sparse_96, Rand1_100* and *Rand2_100* are mapped onto an (8x8) mesh, and *Rand3_100, Rand4_100, Rand1_300, Rand2_300* and *Fppp_334* are mapped onto a (10x10) mesh. Random task-graphs for the same mesh size are chosen to mimic applications with varying degrees of communication, e.g., *Rand1_300* and *Rand2_300* both map to a 100-core mesh, but have very different communication patterns.

The power values of routers and links (32-bit wide) for different voltages, frequencies and router complexities at full load are obtained from ORION 2.0 [19]. The active times of NoC-components are used for computing their energy values,

$$E_{component} = Power_{component} * AT_{component}$$

where active time (*AT)* of a NoC-component is the minimum time period for which the component needs to be active (at full load) in a single traversal of the application (task-graph). Note, the router power values obtained are used for nominal V_T, and are scaled for varying V_T values. As the VLCs and MCFIFOs required to interact between *VI*s incur a power overhead that is proportional to their voltage supply, we consider the power overhead of these components, with the actual power values based on reported overheads from existing literature [20].

5.2 Results

To evaluate the quality of solutions generated by our variation-aware NoC synthesis framework (VERVE), we compare the results with the process variation-aware synthesis framework from [7], which is the best known prior work in recent literature that attempts to solve a similar problem as the one we solve. Note that [7] considers just the shapes of the voltage islands (not the actual locations of *VI*s on the MPSoC) to alleviate the effects of process variations. We also compare our work with a variation-unaware NoC synthesis approach proposed in [21], which is aimed at energy-efficient core-to-tile mapping and routing. We implemented the frameworks presented in [7] and [21] to the best of our understanding and used the same models for all implemented approaches to ensure a fair comparison of the algorithms used. We implemented all frameworks (including VERVE) in C++, and simulated them on a Windows machine with an Intel i5 processor and 6 GB main memory. Note that while computing energy dissipation in our experiments, both dynamic power and static power are considered. Figure 5 shows the comparison of results for compute-energy for the approaches in [21], [7], and VERVE; as well as the *ideal-case* solution assuming no process variations.

For the ideal-case, *VI-placement* (minimizing compute energy) is rendered redundant because all tiles have the same V_T=0.4V; therefore, the loss in energy efficiency and performance due to the effects of process variations are avoided. Further, rated frequency corresponding to each voltage level can be sustained for the respective *VI* in the ideal-case, unlike a realistic situation with process variations where the operating frequency of the *VI* is pulled down to the lowest frequency within that *VI*. From results, it can be observed that our VERVE framework achieves 3.4% improvement in compute energy on average over [7] and 3.8% improvement over [21]. Note that as static and dynamic components of power are affected differently by variation in V_T values, the scope of the compute energy optimization is quite limited; specifically, compute energy of the ideal-case (which is unachievable) is just 8.5% lower (on average) compared to the compute energy of the variation un-aware work of [21]. Thus, even though savings in compute energy over [7] seem modest in absolute terms; with respect to ideal-case (base-case), they are quite significant (up to 52%).

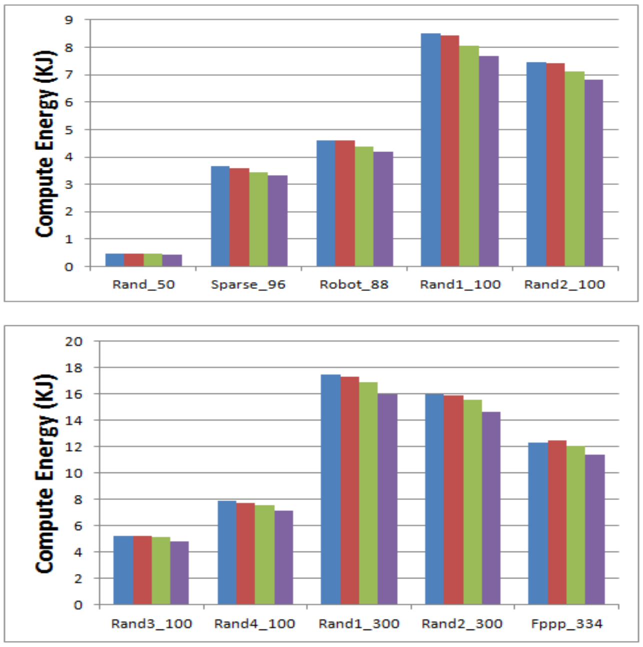

Figure 5: Compute energy (K Joules) comparison between the frameworks from [7], [21], VERVE, and the ideal-case

Table I: Average core operating frequencies (in MHz) for benchmarks with different mesh sizes

mesh-sizes	49	64	100
[21]	1374	1339	1322
[7]	1372	1348	1324
VERVE	1398	1425	1369
ideal-case	1525	1528	1535

Interestingly, the *VI-placement* scheme in VERVE also has a positive effect on average operating frequency of the MPSoC. It turns out that cores with a specific supply voltage are generally affine to tiles with a certain range of V_T values for optimal compute energy. Our *VI-placement* step captures this principle, thus resulting in lower V_T deviation within *VI*s. As a consequence, frequency deviation within each *VI* is also reduced, thus preventing the operating *VI* frequency (minimum frequency of all cores within the *VI*) from being pulled down substantially. Table I shows improvements in average core operating frequency for VERVE over [7] and [21]. A notable increase in average allowed frequency can be observed, up to 5.7% and 6.4%, compared to prior works [7] and [21] respectively.

Next, we present the improvements in communication efficiency in the NoC fabric for the VERVE framework. We found that even though mapping of cores is restricted to the tiles assigned to the same *VI* (after the *VI-placement* step), our core-to-tile mapping technique results in significant reductions (14% on average) in total traffic over [7]. When compared to [21] (where incremental swapping is done without the *VIs* being fixed on the mesh), there is on average, less than 2% increase in total traffic using VERVE. Our routing approach which is aimed at minimizing inter-island link insertions in addition to minimizing worst case (WC) communication overlap of the flows generally results in significantly reduced communication energy.

Table II: Comparison of total number of inter-island links needed between [7], [21], and VERVE

BM	Rand_ 50	Sparse_ 96	Robot_ 88	Rand1_ 100	Rand2_ 100
[21]	47	25	34	78	67
[7]	47	34	56	68	64
VERVE	23	27	47	64	70
BM	**Rand3_ 100**	**Rand4_ 100**	**Rand1_ 300**	**Rand2_ 300**	**Fppp_ 334**
[21]	119	98	105	132	106
[7]	73	76	72	76	73
VERVE	60	59	80	88	81

Table II shows how the total number of inter-island link insertions is optimized in VERVE compared to the framework in [7]. Figure 6 quantifies the energy impact of more efficient inter-island link insertion. VERVE achieves 19% average communication energy savings over [7].

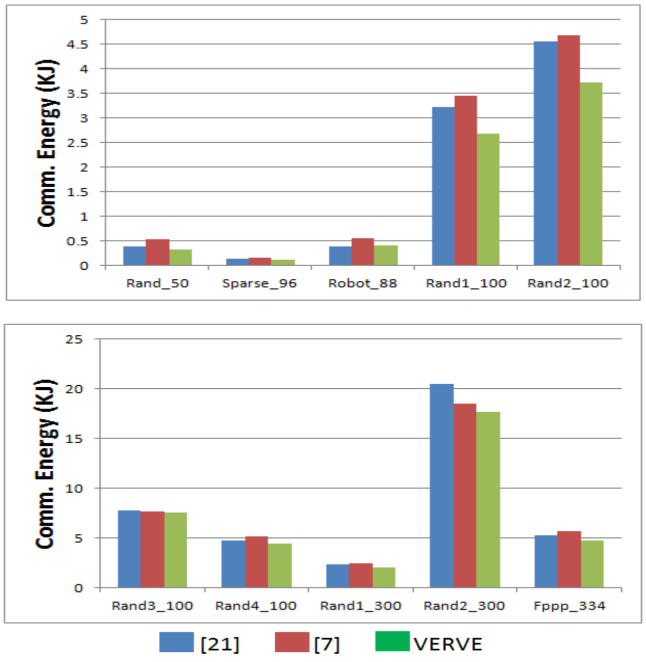

Figure 6: Communication energy (K Joules) comparison between frameworks from [7], [21], and VERVE

Table III: Comparison of execution times (in secs) between [7], [21], and VERVE

BM	Rand_ 50	Sparse_ 96	Robot_ 88	Rand1_ 100	Rand2_ 100
[21]	3.4	2.7	4.6	9.6	12.1
[7]	3.1	2.8	5.7	37.6	55.0
VERVE	1.7	1.6	2.6	6.4	9.5
BM	**Rand3_ 100**	**Rand4_ 100**	**Rand1_ 300**	**Rand2_ 300**	**Fppp_ 334**
[21]	38.6	24.6	14.8	93.3	27.5
[7]	139.9	82.1	41.5	333.2	61.1
VERVE	36.4	20.8	8.1	81.1	19.7

We also observe an average reduction of 12.5% in communication energy using VERVE compared to [21] (as shown in Figure 6). The result can be explained as follows. First, note that routers, MCFIFOs, VLCs operate at the same voltages/frequencies as the cores they are associated with, and are affected by the same variations in threshold voltage; and thus experience similar energy savings as the compute cores. Therefore, savings in compute energy, derived from the *VI-placement* step in VERVE contribute toward minimizing the communication energy as well. Secondly, even though the routing scheme in [21] attempts to minimize the total number of inter-island links, a few *VI*s are spread-out (far from rectangular or circular in shape) with its mapping approach, which uses incremental swapping with no restrictions on *VI* locations. This leads to an increase in the total number of inter-island links inserted, as shown in Table II (compared to VERVE); which in turn increases communication energy. Third, our routing scheme considers the added objective of minimizing the bottleneck communication volume along paths, over [21].

Finally, Table III shows a comparison of execution runtimes of the synthesis frameworks. Given an application-specific core-graph, our decomposed algorithmic approach for mapping cores onto the mesh topology composed of *VI*-to-tile mapping and subsequent core-to-tile mapping using incremental swapping within individual VIs not only provides more energy-efficient results but also takes less simulation time compared to the incremental core-to-tile mapping approach proposed in [7]. Also, the routing step which accounts for the biggest portion of the simulation time, is performed in a much more time-efficient way by VERVE, compared to the routing approach in [7] which uses a genetic algorithm. In [21], incremental swapping is performed on Ω initial mapping instances to optimize the network traffic, whereas in VERVE, incremental swaps are performed on only one initial mapping instance on a restricted search space (all swaps within *VI*s); which leads to better time-efficiency.

6. Conclusion

In this paper we have proposed a novel process-variation aware framework for the system-level synthesis of NoC-based MPSoCs (VERVE) which optimizes both computation and communication energy. We proposed a novel *VI-placement* methodology to utilize the knowledge of process variation distribution over the die to optimize the design for computation energy as well as average operating frequency. In addition, we proposed novel mapping and routing techniques to optimize inter-core communication in the NoC. Our VERVE framework, on average, produces 3.4% savings in computation energy and 19% savings in communication energy compared to the best known prior work on NoC synthesis that also considers process variations.

Acknowledgement

This research is sponsored in part by grants from NSF (CCF-1252500) and SRC.

References

[1] J. Tschanz, K. Bowman, and V. De, “Variation-Tolerant Circuits: Circuit Solutions and Techniques,” Proc. DAC, 2005, pp. 762-763.

[2] E. Humenay, D. Tarjan, and K. Skadron, “Impact of Process Variations on Multicore Performance Symmetry,” Proc. DATE, 2007, pp. 1653-1658.

[3] S. Vangal et al., “An 80-Tile 1.28 TFLOPS Network-on-Chip in 65 nm CMOS,” IEEE ISSCC, 2007, pp. 98–589.

[4] Tilera Corporation. TILE64™ Processor. Product Brief. 2007.

[5] L. Benini, and G. De-Micheli, “Networks on Chip: A New SoC Paradigm,” Computer., vol. 35, no. 1, pp. 70-78, Jan. 2002.

[6] D. Lackey et al., “Managing Power and Performance for Systemon-Chip Designs using Voltage Islands,” Proc. ICCAD, 2002, pp. 195-202.

[7] S. Majzoub, R. Saleh, S. Wilton, and R. Ward, “Energy Optimization for Many-Core Platforms: Communication and PVT Aware Voltage-Island Formation and Voltage Selection Algorithm”, IEEE Trans. Computer-Aided Design, vol. 29, no. 5, pp. 816-829, May 2010.

[8] S. Sarangi et al., “VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects,” IEEE Trans. Semiconductor Manufacturing, vol. 21, no. 1, pp. 3-13, Feb. 2008.

[9] S. Hong, S. Narayanan, M. Kandemir, and O. Ozturk, “Process variation aware thread mapping for Chip Multiprocessors,” Proc. DATE, 2009, pp. 821-826.

[10] S. Garg, and D. Marculescu, “System-Level Process Variation Driven Throughput Analysis for Single and Multiple Voltage-Frequency Island Designs,” Proc. DATE, 2007, pp. 1-6.

[11] S. Herbert, S. Garg, and D. Marculescu, “Exploiting Process Variability in Voltage/Frequency Control,” IEEE Trans. VLSI, vol. 20, no. 8, pp. 1392-1404, Aug. 2012.

[12] F. Wang et al., “Variation-Aware Task and Communication Mapping for MPSoC Architecture,” IEEE Trans. CAD, vol. 30, no. 2, pp.295-307, Feb. 2011.

[13] D. Mirzoyan, B. Akesson, and K. Goossens, “Process-variation aware mapping of real-time streaming applications to MPSoCs for improved yield,” Proc. ISQED, 2012, pp. 41-48.

[14] R. Teodorescu, and J. Torrellas, “Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors,” Proc. ISCA, 2008, pp. 363-374.

[15] U. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, “Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip,” Proc. DAC, 2007, pp. 110-115.

[16] W. Jang, D. Ding, and D. Pan, “A Voltage-Frequency Island Aware Energy Optimization Framework for Networks-on-Chip,” Proc. ICCAD, 2008, pp. 264-269.

[17] http://www.arm.com/products/processors/selector.php

[18] http://www.kasahara.elec.waseda.ac.jp/schedule/

[19] A. Kahng, B. Li, L. Peh, and K. Samadi, “ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration,” Proc. DATE, 2009, pp. 423-428.

[20] T. Chelcea, and S. Nowick, “A Low-Latency for Mixed-Clock Systems,” CSW on VLSI, 2000, pp. 119-126.

[21] N. Kapadia, and S. Pasricha, “VISION: A Framework for Voltage Island Aware Synthesis of Interconnection Networks-on-Chip,” Proc. GLSVLSI, 2011, pp. 31-36.

[22] VARIUS model, http://www.cse.ohiostate.edu /~teodores/ arch/tools/varius/varius.html

[23] D. Starobinksi et al., “Application of network calculus to general topologies using turn-prohibition,” Trans. on Networking, 11, 3, (June 2003), 411-421.