Contents

1 Introduction

The emergence of Network-on-Chip (NoC) architectures has revolutionized the design of complex systems-on-chip (SoCs) by offering scalable, efficient, and high-performance communication infrastructures. NoCs replace traditional bus-based interconnects with a mesh of interconnected processing elements (PEs), providing a flexible platform for communication between various components of a system. As the demand for computational power continues to escalate in modern computing systems, particularly in applications such as artificial intelligence, multimedia processing, and high-performance computing, the significance of NoC based architectures becomes increasingly pronounced.

In parallel, the representation and execution of applications have been formalized through task graphs, which capture the dependencies inherent in computational tasks. Task graphs provide a visual abstraction of the workflow, enabling efficient task scheduling and resource allocation strategies. The optimization of task graph execution on NoC-based architectures presents a multifaceted challenge, requiring the orchestration of computation and communication resources to minimize latency, maximize throughput, and ensure balanced resource utilization.

Scheduling of task graphs on the processing platform can be classified mainly into two types (i) static/offline scheduling, and (ii) dynamic/online scheduling. In static scheduling, all details regarding the task graph are known in advance. So tasks to processors allocations, start times of tasks, and the schedule for routing messages are pre-computed and are loaded onto the processing platform. In contrast, dynamic/online scheduling does not have complete information about the task graph at the beginning, and decisions regarding task allocation and scheduling are made during runtime. Static scheduling is preferred where there is a need for predictability and dynamic scheduling usually has higher performance gains.

In NoC architectures, the on-chip communication infrastructure is organized as various types of networks such as crossbar, star, ring, tree, 2D mesh, and 2D torus. In this work, we focus on $n \times n$ two-dimensional (2D) rectangular mesh network. 2D rectangular mesh network typically comprises a mesh where each processing element (PE) is connected to its own router and routers are connected by communication links. These routers act as the junction for the transmission of data packets. The use of NoC enables us for an efficient communication infrastructure that avoids the possibility of bottleneck performance which makes its on-chip communication better than fully connected or shared bus architecture.

Scheduling a task graph on NoC architecture involves mainly two phases (i) selection of PEs for the tasks, (ii) routing of message flits between different tasks scheduled on different PEs. The communication infrastructure of NoC can be taken to our advantage for routing of message flits for faster data transmission between tasks which eventually helps in faster completion times of tasks. However, the problem of routing of message flits in task graph in NoC architecture is an NP-complete problem, and determination of optimal routing of message flits in NoC architecture involves exploration of large state space which is often difficult for large sized $n \times n$ rectangular mesh networks. Therefore employing an effective scheduling strategy that provides good solutions with lower time complexity is required.

In this work, we propsose a hueristic scheduling strategy that facilitate the the concept of gradual expansion, wherein task graphs scale their footprint within the

NoC to accommodate additional computational tasks or adapt to changing resource availability. This approach not only optimizes resource utilization but also enhances system capability to workload variations.

Moreover, list-based scheduling algorithms play a pivotal role in optimizing task execution sequences and resource utilization. List-based scheduling methods, such as List Scheduling and Modified List Scheduling, prioritize tasks based on their dependencies and available resources, resulting in efficient task execution sequences that minimize overall makespan and resource idle time. By leveraging list-based scheduling techniques within NoC architectures, we aim to further enhance the performance and scalability of task graph execution, particularly reducing the overall makespan.

2 Literature Survey

The process of mapping tasks onto the NoC platform involves determining the optimal processing element of tasks inside the platform, taking into consideration optimization factors such as minimizing energy consumption, reducing total execution time, etc.

Task graph scheduling within Network-on-Chip (NoC) architectures, managing inter-task messages efficiently is crucial for optimizing system performance. Inter-task messages represent data exchanges between computational tasks, influencing task execution and resource utilization. Routing strategies within the NoC determine the path of inter-task messages, impacting makespan, throughput, and system efficiency. Various routing algorithms, including deterministic and adaptive schemes, have been proposed to address the challenges of inter-task communication in NoC environments. Integrating communication-aware task scheduling techniques with routing algorithms enables holistic optimization of communication patterns, reducing contention and improving message delivery times.

Traditionally, task mapping on heterogeneous computing platforms has been dealt with static list based heuristic strategies. Many useful list based strategies have been developed over the years [4], *HEFT, CPOP* [15], *PEFT* [16], etc. HEFT prioritizes tasks based on upward rank values to minimize earliest finish times, while CPOP considers both upward and downward rank values, focusing on critical tasks to minimize total execution time. Through rigorous comparison studies, including randomly generated and real application graphs, the algorithms demonstrate superior performance in terms of schedule quality and cost, showcasing improvements in schedule length ratio, speedup, frequency of best results, and average scheduling time. 20 list based heuristics have been compared in [5] and decided that *HEFT* performs better interms of makespan and robustness

In NoC communication architecture, routing algorithms can be classified into two types (i)oblivious routing, and (ii)adaptive routing. In oblivious routing, network traffic is routed based on a predetermined, fixed routing scheme that does not take into account the current network conditions or congestion. Adaptive routing, on the other hand, is a dynamic routing approach that takes into account real-time network conditions, such as congestion, link failures, and other factors. In [9], authors review various XY oblivious, adaptive routing algorithms, which are widely used in NoC due to their simplicity. These XY routing strategies are classified based on the specific requirements and environmental considerations they address. The

IX/Y routing algorithm aims to minimize network collisions, DyXY routing focuses on preventing deadlocks and livelocks, XYX routing offers fault tolerance, and the Adaptive XY routing algorithm optimizes network resource utilization. This study provides insights into the diverse XY routing options available for efficient on-chip communication in NoC-based systems.

[17], proposes a novel list-scheduling algorithm, Communication-aware Predictive Priority Task Scheduling (CPPTS), tailored for heterogeneous multi-processor systems-on-chips (MPSoCs) based on Network-on-Chip (NoC) architectures. By combining heterogeneous system list scheduling with NoC-based mapping and scheduling algorithms, CPPTS leverages prediction and communication-awareness to prioritize tasks and select processors efficiently. Experimental results, encompassing both randomly generated Directed Acyclic Graphs (DAGs) and real-world applications like Gaussian elimination and Cybershake, demonstrate CPPTS's superior performance over five other algorithms across various routing methods and system structures. Notably, CPPTS achieves improved scheduling results while maintaining quadratic time complexity, highlighting its effectiveness in enhancing execution efficiency for applications on NoC-based heterogeneous MPSoCs.

In [11], authors introduce an approach that combines both design-time planning and run-time adaptability. During design, the system generates different allocation templates based on factors like processor capabilities, communication energy, and task timing using particle swarm optimization(PSO) and stores them in a special node called platform manager. During run time, the system adjusts these allocations based on available resources and application requirements thus making it flexible for dynamic workloads. However, the routing of inter-task messages is through a table-based routing algorithm, which can be improved to use the NoC communication architecture more efficiently.

The main problem caused due to the above-mentioned types of XY algorithms are congestion and faults which cause a delay in transmission, which degrades the performance of the NoC. So, A congestion aware algorithm is developed in [12] which mainly works by sending the congestion information i.e., the percentage of input buffers occupied to the neighbor routers along with the data flits and the congestion information of neighbor routers are stored in a table. The router's routing decision from the source to the destination relies on the congestion information received from neighboring routers. In [7], authors mainly explore fault-tolerant routing algorithms in 2D mesh networks and propose innovative algorithms capable of maintaining system performance in the presence of faults, the suggested algorithms prioritize the use of shortest paths whenever possible, strategically employing non-minimal paths when necessary.

The above works are developed with a routing algorithm that only looks at the network state at that moment and doesn't look ahead in state space for routing decisions. [10] introduces a heuristic Dynamic Spiral Mapping (DSM) algorithm tailored for 2-D mesh topologies. The research explores two distinct approaches derived from DSM: Full Dynamic Spiral Mapping (FDSM) and Partial Dynamic Spiral Mapping (PDSM). The primary focus of the investigation is the comparison of reconfiguration times between FDSM and PDSM as a metric to evaluate the algorithm's efficacy. In [2], a novel and fully adaptive routing algorithm named HRA (Heuristic-based Routing Algorithm) is introduced, drawing inspiration from the A-star search algorithm. This algorithm calculates score f for every node by adding up

two factors, the minimum number of hops from the present node to the destination node and the percentage of occupied space in the neighbor input buffer. HRA Algorithm is designed to operate in 2D-mesh NoC architectures, offering a distributed, congestion-aware, and fault-tolerant solution by leveraging only the local information available at each router's neighbors. Notably, HRA takes a proactive approach to preventing deadlock situations by intelligently avoiding 2-node and 4-node loops. Results presented in this paper demonstrate the efficiency of HRA, showcasing its remarkable reliability even in the presence of numerous faulty links and its capacity to deliver low latencies across diverse NoC sizes, underscoring its potential to significantly enhance the overall performance of NoC systems.

In [8], authors present a novel packing strategy designed for run-time mapping of application tasks on NoC-based Heterogeneous Multiprocessor Systems on Chip (MPSoCs). The objective of the proposed strategy is to identify and utilize available free resources efficiently, aiming to minimize task mapping time and strategically position communicating tasks in close proximity. Similarly in [13], the authors present a collection of communication-aware runtime mapping heuristics specifically designed for efficiently mapping numerous applications onto a Multiprocessor System on Chip (MPSoC) platform, where each processor element (PE) is capable of handling multiple tasks. The novel mapping heuristics methodically assess the accessible resources, giving priority to assigning neighboring communicative tasks to the same processing element (PE). Significantly, the suggested heuristics prioritize jobs inside a single application that is nearby, with the goal of further minimizing communication overhead.

In conclusion, the literature review presents a comprehensive overview of task graph scheduling and routing algorithms within Network-on-Chip (NoC) architectures. The review highlights the importance of efficient task mapping and inter-task communication management in optimizing system performance. Various scheduling algorithms, such as HEFT, CPOP, and CPPTS, demonstrate superior performance in terms of schedule quality and cost across heterogeneous computing platforms. Additionally, routing algorithms play a crucial role in enhancing overall system reliability. The exploration of innovative approaches, such as DSM, HRA, underscores ongoing efforts to address workload requirements and optimize resource utilization in NoC-based systems. Overall, the literature review underscores the significance of holistic optimization strategies that integrate task scheduling, inter task message routing to meet the evolving demands of modern computing applications.

3 Research Gap, Motivation and Scope

The existing literature survey highlights several noteworthy contributions in the field of scheduling task graphs on various on-chip communication infrastructures. However, a critical research gap exists in addressing the exploration of novel heuristic mapping strategies and routing challenges of inter-task messages within a NoC architecture.

Furthermore, while considerable research has been conducted on task graph scheduling algorithms within NoC architectures, there remains a notable gap in the exploration of mapping strategies for task allocation that facilitate the the concept of gradual expansion, where task graphs scale their footprint within the NoC. Additionally, the integration of clustered mapping with heuristic message scheduling

algorithms remains largely unexplored, limiting our understanding of its impact on overall system performance and scalability.

The literature review highlights the prevalence of oblivious and adaptive routing algorithms in NoC communication architecture, with significant importance on XY routing strategies. While these algorithms address specific requirements such as minimizing collisions, preventing deadlocks, ensuring fault tolerance, there remains a gap in the exploration of scheduling strategies that can rightly anticipate by navigating through the state space for routing decisions. The motivation stems from the need for more proactive and heuristic routing algorithms.

In the evolving workloads in multi-core computer systems, having a rigid systematic strategy for task mappings, routing the inter-task message can lead to transmission delays and performance issues. While some existing works, such as the one presented in [11], combine design-time planning and run-time adaptability, they still employ table-based routing algorithms. This presents an opportunity for improvement, particularly in using the capabilities of NoC communication architecture more efficiently.

Addressing this research gap presents an opportunity to develop novel task mapping strategies that leverage the inherent parallelism and communication capabilities of NoC architectures, thereby optimizing resource allocation and execution efficiency. This research aims to bridge this gap by proposing a novel approach that combines novel mapping techniques with heuristic scheduling algorithms, ultimately contributing to the advancement of task graph scheduling in NoC-based systems.

Motivated by the challenges posed by congestion, transmission delays, and performance degradation in NoC systems, the proposed research aims to contribute to the field by developing better scheduling strategies for inter-task message routing in NoC architectures. The motivation stems from the need for more proactive and efficient routing algorithms that leverage local information to enhance congestion awareness and reduce delays and overall system performance.

The scope of this research is to solve the identified gap by exploring various heuristics for task mapping as well as inter task communication in NoC architectures different scheduling techniques for routing inter-task messages in a mesh NoC. Unlike previous approaches, which often rely on rigid mapping and routing strategies, the focus here is more developing heuristic based strategies that consider both the network state at the current moment and look ahead in the state space for more informed routing decisions.

4 System Model

The system model associated with this work is presented by describing the platform, computation model, and assumptions.

4.1 Platform

The platform model is a heterogeneous $n \times n$ 2D rectangular mesh Network-on-Chip (NoC) with homogenous links, where n represents the mesh size. $n \times n$ NoC consists of n^2 nodes, each consisting of a processing element (PE) and its dedicated router. The router is responsible for communicating and transmitting the data to the neighboring routers. This mesh topology enables direct communication to neighboring

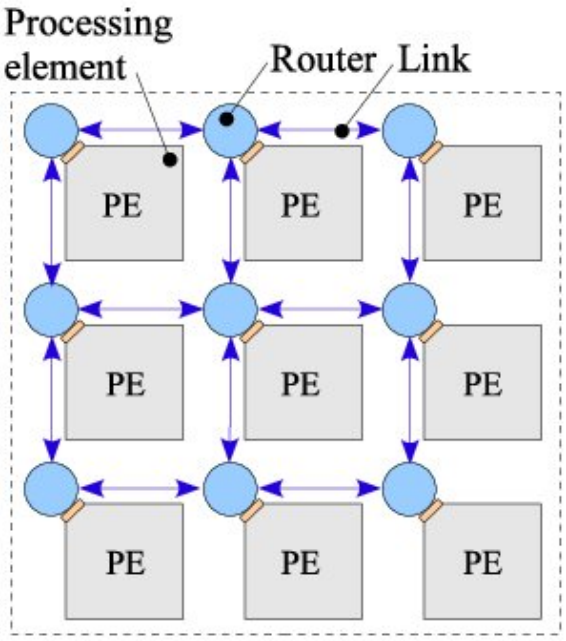


Figure 1: Heterogeneous 2D Mesh NoC

routers i.e., north, east, south, and west. Data to be transmitted through the router can come from its PE or any neighboring router for further transmission.

Since, processors are heterogenous, task τ_i takes different execution times of different processors. τ_i takes $\omega_{i,k}$ for execution on kth processor which is given in matrix W of size $|V|\times \mathrm{n}^2$.

Fig 2 shows the router architecture consisting of input ports, output ports, and a switch. Every router has 5 input ports (north, east, south, west, local) where the local input port is connected to its own PE and north port is connected to the south output port of the north neighbor, and so on. Each port consists of a buffer which is a queue of flits, these flits are routed using the switch sequentially using a switch to its target output port.

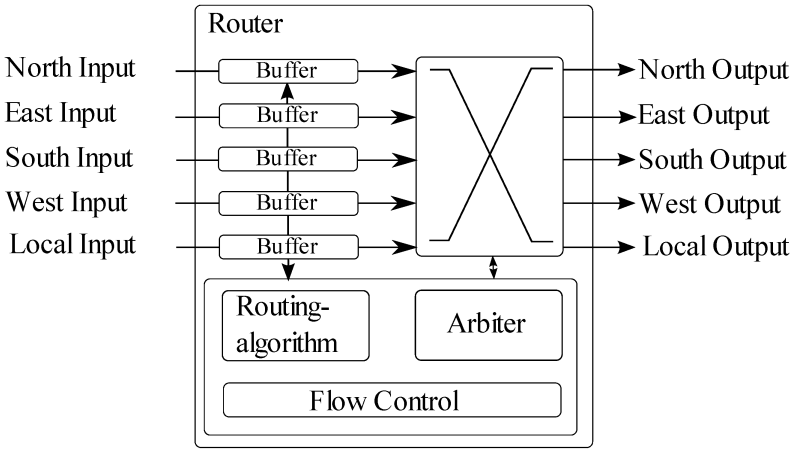


Figure 2: Router Architecture

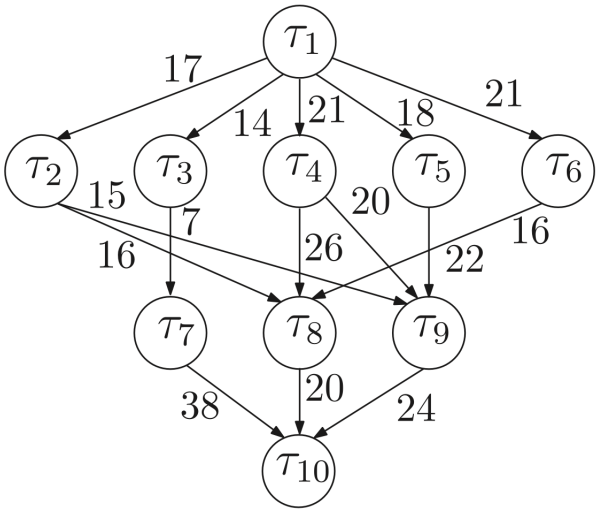


Figure 3: Task Graph

4.2 Computation Model

An application task graph considered in this work is represented by a directed acyclic graph *G(V, E)*, where each node of the graph represents the task τ_i, an edge represents the message sent from τ_i to τ_j. we define m_{ij} as a message sent from τ_i to τ_j in k flits, the value of m_{ij} is the size of the message. These inter-task messages are to be routed on the selected platform.

4.3 Assumptions

1. DAG *G(V, E)* has a single entry task (node) τ_i and a single exit task (node) τ_n. If the input DAG contains multiple source/sink nodes, we add a single dummy source/sink task (node).
2. The order of message flits of a message at the destination router doesn't matter.
3. Size of the input port buffers is infinite.
4. We consider the sequential transmitting of messages.
5. Each message flit takes 1 unit time for each hop.

Example: Fig. 1 shows a 3×3 2D rectangular mesh NoC consisting of 9 processing elements and 9 routers. Fig. 3 shows an DAG consisting of 10 tasks $\{\tau_1, \tau_2, ..., \tau_{10}\}$, 15 messages $\{m_{12}, m_{13}, m_{14}, ..., m_{910}\}$ where edge weight represent the message size like $e_{12} = 17$, $e_{13} = 14$, etc.

5 Problem Statement

CPS application is modeled as task graph(DAG) *G(V, E)* is given and is to be scheduled on a $n \times n$ heterogeneous 2D rectangular mesh NoC computing platform. The overall objective is to generate the processor-to-task mappings along with their start times and end times, and port schedules for all the n^2 routers in the network.

Generating the routers schedule will indirectly give the path for all inter-task messages. Minimizing the overall schedule length is the ultimate goal in scheduling this task graph.

Minimizing the schedule length is equivalent to minimizing the effective finish time of the sink node of the task graph. Effective finish time $EFT(\tau_i, PE_n)$ is defined for a task, PE pair which represents the finish time of τ_i if it is mapped to PE_n as shown in equation 1. $EST(\tau_i, PE_n)$ represents the earliest start time τ_i can start its execution on PE_n. $\omega_{i,n}$ is the time PE_n takes to execute τ_i after start, which is already known. To calculate the $EST(\tau_i, PE_n)$, which depends on the transmission times of its predecessor messages. so, to minimize the transmission times of messages, an heuristic routing algorithm that takes the routing decision based on network conditions after exploring state space is crucial which is the objective.

$$EFT(\tau_i, PE_n) = EST(\tau_i, PE_n) + \omega_{i,n} \tag{1}$$

The message m_{ij} can be represented by a list $m_{ij} = [m_{ij}^1, m_{ij}^2, ..., m_{ij}^k]$. To say m_{ij} is successfully transmitted from τ_i to τ_j, every flit in the list needs to reach the PE that is mapped to τ_j. Time taken for the transmission of message m_{ij} is $t_{m_{ij}} = max(t_{m_{ij}^1}, t_{m_{ij}^2}, ..., t_{m_{ij}^k})$. So, it is required to find the path for every flit in such a way that it reaches the destination PE in the minimum time possible.

Problem Statement: *Given a task graph application DAG G(V, E), Execution time matrix W, NoC platform of size n, the objective is to generate the processor-to-task mappings along with their start times and end times, and port schedules for all the n^2 routers in the network, minimising the makespan of the schedule.*

6 Proposed Heuristics

In this section, we introduce Communication aware List Scheduling algorithm **(CLS)**. This whole section is divided into two parts, in 6.1, we introduce a novel list based strategy for processor selection and in 6.2 routing of inter-task messages in NoC is discussed.

6.1 List based processor allocation

This phase starts with creating a task priority list of the given DAG, emphasizing the importance of task rank calculation in optimizing task allocation within the NoC architecture. Task ranks serve as crucial indicators of task priority, guiding their assignment to processing elements. Task ranks are determined based on precedence constraints and average execution time. Specifically, tasks are prioritized based on their rank values, which reflect their importance in the task graph's execution sequence. Tasks with higher rank values are considered and prioritized accordingly for allocation to processing elements. This rank calculation strategy ensures efficient utilization of processing resources and facilitates the timely execution of critical tasks within the system.

Rank of a task

$$rank(\tau_{exit}) = \overline{ET_{exit}} \tag{2}$$

$$rank(\tau_i) = \overline{ET_i} + \max_{\tau_j \in \text{succ}(\tau_i)} (rank(\tau_j) + CC_{i,j}) \tag{3}$$

$\overline{ET_i}$ = average execution time of task τ_i on all processors of given NoC platform.
$CC_{i,j}$ = Message size of message from τ_i to τ_j

task_priority_list is generated by arranging the tasks based on their computed rank values in non increasing order, we iterate through the list sequentially and allocate the processor to that task based on the *Effective Finish Time* (EFT) value.

***Effective Finish Time* (EFT)**: This value is given for a task-processor τ_j, PE_n pair, which is the finish time of execution of task τ_j if alloted PE_n

$$EFT(\tau_j, PE_n) = EST(\tau_j, PE_n) + \omega_{i,n} \tag{4}$$

***Earliest Start Time* (EST)**: This value is given for a task-processor τ_j, PE_n pair, which is the earliest time at which τ_j can start its exectution on PE_n, if allocated.

$$EST(\tau_j, PE_n) = max[avail(PE_n), \max_{\tau_i \in \text{pred}(\tau_j)}(EFT(\tau_i) + CC_{i,j})] \tag{5}$$

Heuristic - 1 : In this novel heuristic for processor allocation, we adopt a strategy that optimizes task allocation by limiting the candidate processors for consideration. Specifically, instead of evaluating every processor within the NoC architecture, we focus solely on processors that are at most one hop away from at least one of the source processors. This approach is supported by the motive that the task graph should expand in a gradual way in the NoC and this also approach significantly reduces the search space for processor selection, allowing for more efficient allocation decisions. By considering only processors in close proximity to the source processors, we prioritize communication proximity and minimize data transfer latency, thus enhancing overall system performance. This heuristic ensures that tasks are allocated to processing elements that can execute them with minimal communication overhead, facilitating faster task completion and improved system efficiency.

6.2 Routing of inter-task messages

Heuristic - 2 : In this section, we propose a routing heuristic that prioritizes the consideration of shortest routes for inter-task message transmission within the NoC architecture. Our approach focuses on evaluating the earliest finish times values for the shortest paths, determined by manhattan distance, to minimize message transmission latency and ensure efficient communication. Additionally, we introduce a customizable parameter that allows users to specify the maximum number of shortest routes to explore. This flexibility empowers system designers to tailor routing decisions according to specific performance objectives and constraints, striking a balance between communication efficiency and computational overhead. Through this adaptable heuristic, we aim to optimize message delivery times while offering versatility in routing strategy selection within NoC-based systems.

6.3 Algorithm

schedule_task_graph(), outlines the task graph scheduling process on the NoC platform. Firstly, the priority of each task is computed based on its importance in the

graph. Then, tasks are organized into a list sorted by priority. For each task, a candidate list of processors is generated based on availability. Tasks are tentatively assigned to processors, and message scheduling is performed if the task has dependencies. Finally, tasks are allocated to the processor with the minimum finish time, and the actual start and end times are recorded.

ALGORITHM 1: *scheule_task_graph*()

```
Input: DTG G, Platform model P
Output: Schedule of tasks and messages of G on P
Compute the task priority of every task in G;
Create a Task_List consisting of the task nodes sorted in non-increasing order of their
  priorities;
For each processor P_r ∈ P and each R_t ∈ R, create availability list Avail_P_r and Avail_R_t;
while Task_List is non-empty do
    Select the first task T_j from the Task_List;
    if T_j is the entry task then
        for each processor P_r ∈ P do
            Tentatively assign T_j on P_r;
            compute FT(T_j, P_r);
    else
        for subset of processors P_r do
            Tentatively assign T_j on P_r;
            message_list_j = create_message_list(τ_j);
            for each m_{i,j} ∈ message_list_j do
                message_FT_max = schedule_message(P, message_list_j);
            compute FT(T_j, P_r);
    Actually allocate T_j on that P_r on which FT(T_j, P_r) is minimum;
    Insert the tuple (ST[T_j], FT[T_j]), representing the actual start and end times of T_j into
      the sorted list Avail_P_r;
    Remove T_j from the Task_List;
```

create_message_priority_list(), generates a list of messages associated with a given task in ascending order of their ranks. The rank of each message is computed based on the finish time of its predecessor task.

ALGORITHM 2: *create_message_priority_list*()

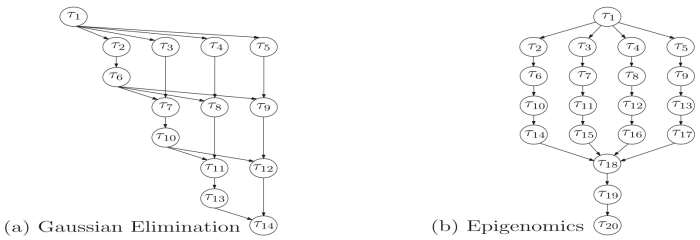
```
Input: DTG G, Task τ_j
Output: List of messages in ascending order of their ranks
for each τ_i in pred(τ_j) do
    rank(m_{i,j}) = FT(τ_i);
return ascending order of m_{i,j} based on their ranks;
```

schedule_message(), schedules the transmission of messages within the NoC platform. For each flit of the message, the algorithm computes the finish time considering different routing routes. The route with the minimum finish time is selected, and the flit is transmitted accordingly. The algorithm returns the maximum finish time among all flits, indicating the completion time of the message transmission.

ALGORITHM 3: *schedule_message*()

Input: message $m_{i,j}$, DTG G, NoC Platform

```
for each flit f_i do
    for route in shortest_routes do
        compute FT(f_i) ;
        if FT(f_i) ≤ FT_min(f_i) then
            min transmission route = route;
    route f_i using min transmission route;
    FT_min = max_∀f_i FT(f_i);
return FT_min;
```



(a) Gaussian Elimination (b) Epigenomics

Figure 4: Benchmark Taskgraphs

7 Performance Evaluation

7.1 Experimental Setup

In the experimental section, we evaluate the proposed algorithms through simulation based experiments using two real world benchmark task graphs: (i) Gaussian Elimination[14], (ii) Epigenomics [3] have been used.

7.1.1 Task Graph Generation

Gaussian Elimination is a task graph representing the process of solving a system of linear equations using Gaussian elimination, a fundamental operation in computational linear algebra. The graph consists of nodes representing mathematical operations such as matrix multiplication, addition, and elimination steps, interconnected by edges denoting data dependencies between tasks. This task graph consists of $((s^2+s-2)/2)$ nodes and (s^2-s-1) edges, where s is the matrix size. Experiments have been conducted task graphs of matrix sizes [4, 8, 12, 16] which translates to [9, 35, 77, 135] tasks respectively.

Epigenomics task graph is derived from real-world bioinformatics applications and represents the analysis of epigenetic data, which plays a crucial role in understanding gene regulation and disease mechanisms. This task graph comprises nodes representing computational tasks involved in processing and analyzing epigenomic data, including sequence alignment, feature extraction, and statistical analysis, interconnected by edges representing data dependencies and workflow constraints. This task graph consists of $4b+4$ nodes and $5b+2$ edges. Experiments have been conducted with branching factors [4, 8, 12, 16] which translates to [20, 38, 52, 68] tasks respectively.

Examples of gaussian elimation DAG with matrix size 5, epigenomics DAG of branching factor 4 has been shown in Figure 4a, 4b respectively.

Gaussian Elimination and Epigenomics task graphs have been created for different parameter values, heterogeneity (β), and Communication-to-Computation Ratio (CCR). β serves as the measure of variability and determines the degree of asymmetry in the execution timings of a task on different processors. CCR is the ratio of the time overhead associated with message transmission and task execution.

7.1.2 Simulation Framework

The simulation framework is implemented in the C++ programming language and runs on a system with the specified configuration: (i) Apple M1 Pro Chip (ii) 16 GiB memory (iii) MacOS Ventura 13.3.1

7.2 Performance metrics

Speedup : Speedup is defined as the ratio of the time it takes to execute a task sequentially compared to the time it takes to execute the same task in parallel (makespan). Consequently, a higher speedup corresponds to a superior scheduling technique. The sequential execution time is determined by allocating all jobs to a single processor, minimizing the overall computational cost of the task graph. The formula for calculating speedup is as follows:

$$Speedup = \frac{min_{p_j \in P}\{\sum_{\tau_i \in T} \omega_{i,j}\}}{makespan} \tag{6}$$

Here numerator denotes the minimum time taken if the tasks are executed sequentially, denominator denotes the makespan achieved by the proposed scheduling algorithm.

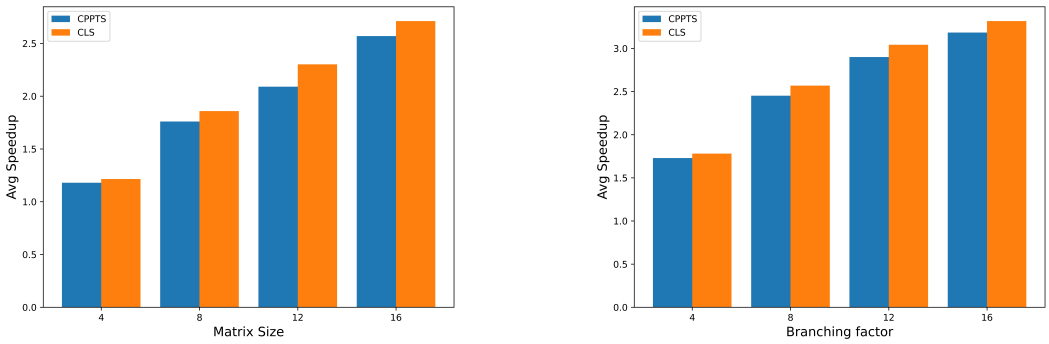
Runtime : Run time for the solution of a task graph scheduling process refers to the time it takes for the scheduling algorithm to determine the schedule. We have considered average generation time of solution as metric.

7.3 Experimental Results

In this section, we present results and comparison with *CPPTS* [17] scheduling strategy, which out performs algorithms like *HEFT* [15], *PEFT* [1], *PPTS* [6]. Experiments have been conducted fixing some system parameters and varying other parameters and averaging over 30 task graphs.

Experiment - 1 *(Impact on speedup varying the input size)*: We investigated the impact of varying the matrix size for Gaussian Elimination and the branching factor for Epigenomics on the performance of the scheduling and routing algorithms. Specifically, we varied the matrix sizes for Gaussian Elimination and the branching factor for Epigenomics in the range [4, 8, 12, 16], while maintaining fixed values for the Communication-to-Computation Ratio (CCR) and β as 1 and 0.5, respectively.

Our experimental results demonstrate that our proposed algorithm CLS consistently outperforms CPPTS, a state-of-the-art scheduling algorithm, across all tested matrix sizes and branching factors. Specifically, we observed around 5% reduction in makespan compared to CPPTS, highlighting the superior performance of CLS in



(a) Gaussian Elimination

(b) Epigenomics

Figure 5: Avg Speedups for varying matrix sizes, branching factors

task graph scheduling within NoC architectures. Furthermore, the graphical analysis of speedup as shown in Figure 5 revealed notable improvements achieved by our algorithm, particularly as matrix size or branching factor increased.

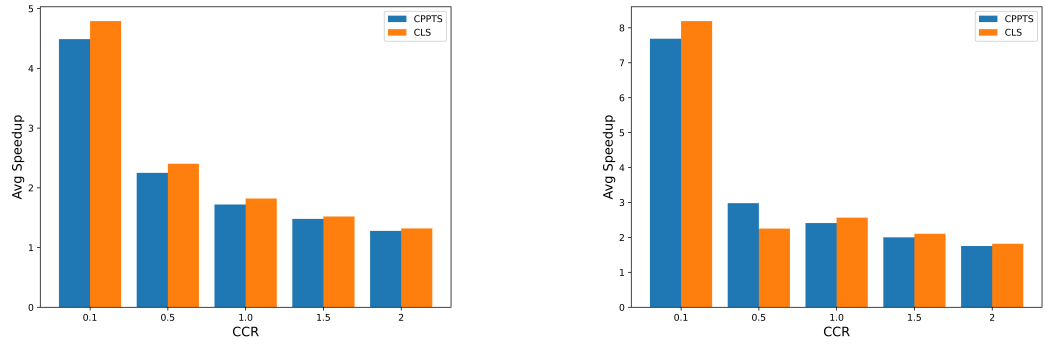
Experiment - 2*(Impact on speedup varying CCR values)*: In this experiment, we explored the impact of varying the Communication-to-Computation Ratio (CCR) values between the range [0.1, 0.5, 1, 1.5, 2] for both Gaussian Elimination and Epigenomics, while maintaining a fixed matrix size of 8 for Gaussian Elimination and a branching factor of 8 for Epigenomics. Additionally, the value of β was fixed at 0.75 to ensure consistency across experiments.

Our experimental findings as shown in Figure 6 revealed that our proposed Communication-Aware List Scheduling Algorithm (CLS) consistently outperformed CPPTS, across all tested CCR values. Notably, even with varying communication-to-computation ratios, CLS demonstrated superior performance in terms of makespan reduction compared to CPPTS.

Moreover, the analysis of speedup metrics indicated a consistent trend across experiments, showing that as the CCR values increased, the speedup achieved by CLS decreased. This observation aligns with expectations, as higher communication-to-computation ratios typically result in increased contention for communication resources, leading to reduced overall system efficiency and slower task completion times.

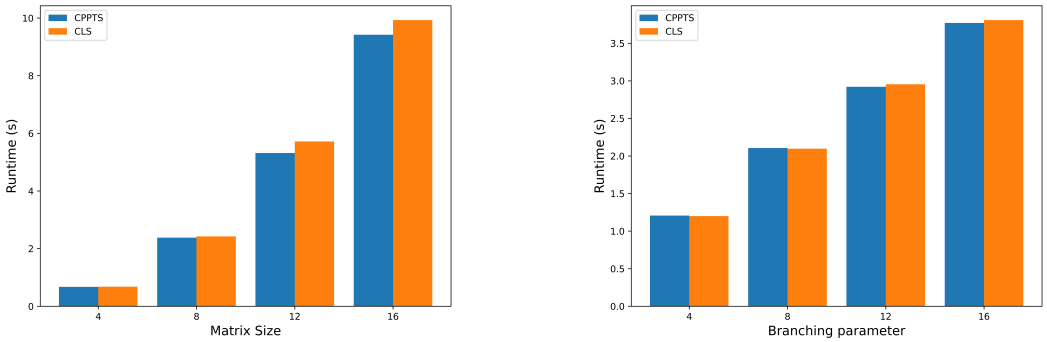
Experiment - 3*(Impact on runtime varying the input size)*: In this experiment, we investigated the relationship between solution generation time and matrix sizes for Gaussian Elimination, branching factors for Epigenomics, while maintaining fixed values for the Communication-to-Computation Ratio (CCR) and β at 1 and 0.5, respectively. The matrix sizes, branching factors were varied in the range [4, 8, 12, 16] to assess the scalability and efficiency of the scheduling algorithm.

Our experimental results as shown in Figure 7, revealed that the solution generation time for the Communication Aware List Scheduling Algorithm (CLS) was slightly higher in some cases compared to CPPTS. Although the difference was very minimal, it was observed that CLS incurred a slightly longer solution generation time. This discrepancy can be attributed to the exploration of a larger state space during solution generation by CLS, which involves considering multiple possible task allocations and routing decisions to optimize the scheduling outcome.



(a) Gaussian Elimination (b) Epigenomics

Figure 6: Avg Speedups for varying Communication-to-Computation Ratio (CCR)



(a) Gaussian Elimination (b) Epigenomics

Figure 7: Avg Runtime for varying matrix sizes, branching factors

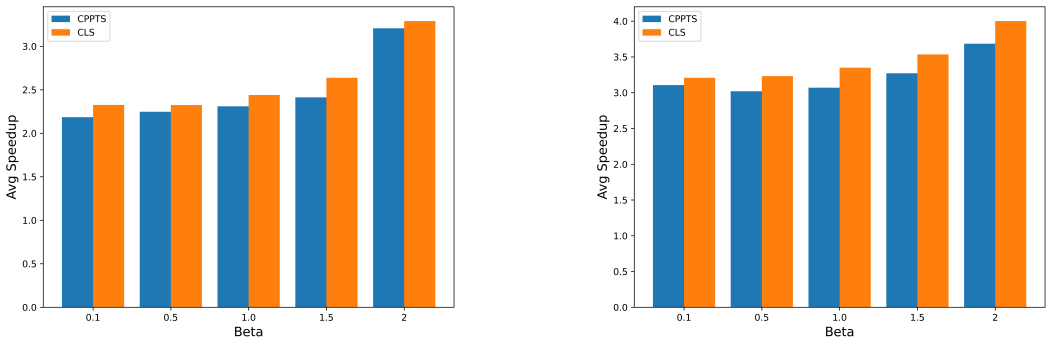
It's worth noting that despite the slightly higher solution generation time, CLS still demonstrated superior performance in terms of makespan reduction and speedup compared to CPPTS, as observed in previous experiments. This indicates that the additional computational overhead incurred during solution generation by CLS is justified by the improved scheduling outcomes and overall system efficiency achieved during task execution.

Experiment - 4*(Impact on speedup varying beta(β) values)*:

In this experiment, we examined the influence of varying the β parameter while keeping the Communication-to-Computation Ratio (CCR) constant at 0.5, We maintained a fixed matrix size of 8 for Gaussian Elimination and a branching factor of 8 for Epigenomics to ensure consistency across experiments.

Our experimental results as shown in Figure 8, with our proposed Communication-Aware List Scheduling Algorithm (CLS) consistently surpassed CPPTS across all tested β values. Despite the variation in β, CLS demonstrated superior performance in terms of makespan reduction compared to CPPTS, indicating its robustness and effectiveness in optimizing task graph scheduling within NoC architectures.

Furthermore, the analysis of speedup metrics consistently demonstrated that as β values increased, the speedup achieved by CLS also increased. This observation



(a) Gaussian Elimination

(b) Epigenomics

Figure 8: Avg Speedups for varying beta(β) values

indicates that higher values of β resulted in improved performance in terms of task completion times and overall system efficiency.

8 Conclusion

In this work, we proposed a novel Communication aware List Scheduling algorithm (CLS) for optimizing task graph scheduling in Network-on-Chip (NoC) architectures. Our approach addresses the critical research gap in exploring heuristic mapping strategies and routing challenges of inter-task messages within NoC architectures. By combining innovative mapping techniques with heuristic scheduling algorithms, CLS offers significant advancements in task graph scheduling, paving the way for improved system performance and scalability.

Through a comprehensive literature review, we identified the limitations of existing scheduling algorithms and the pressing need for more proactive and efficient routing algorithms in NoC systems. Motivated by these challenges, our research aimed to develop better scheduling strategies for inter-task message routing to enhance congestion awareness and reduce delays.

The proposed heuristics, including list-based processor allocation and routing of inter-task messages, offer practical solutions like gradual expansion of task graph on platform, exploring the state space for better routing decisions in NoC architectures. By prioritizing communication proximity and minimizing data transfer latency, CLS ensures faster task completion and improved system efficiency.

Experimental results demonstrated the superiority of CLS over state-of-the-art scheduling algorithms, with significant reductions in makespan and notable improvements in speedup across various input sizes. These findings underscore the efficacy of CLS in addressing the challenges of task graph scheduling in NoC architectures, highlighting its potential for widespread adoption in future computing systems.

References

[1] Hamid Arabnejad and Jorge G Barbosa. "List scheduling algorithm for heterogeneous systems by an optimistic cost table". In: *IEEE transactions on parallel and distributed systems* 25.3 (2013), pp. 682–694.

[2] Asma Benmessaoud Gabis et al. "Heuristic Based Routing Algorithm for Network on Chip". In: Sept. 2016, pp. 39–45. DOI: 10.1109/MCSoC.2016.43.

[3] Shishir Bharathi et al. "Characterization of scientific workflows". In: Dec. 2008, pp. 1–10. DOI: 10.1109/WORKS.2008.4723958.

[4] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm". In: *2010 18th Euromicro conference on parallel, distributed and network-based processing*. IEEE. 2010, pp. 27–34.

[5] Louis-Claude Canon et al. "Comparative evaluation of the robustness of dag scheduling heuristics". In: *Grid Computing: Achievements and Prospects* (2008), pp. 73–84.

[6] Hamza Djigal, Jun Feng, and Jiamin Lu. "Task scheduling for heterogeneous computing using a predict cost matrix". In: *Workshop Proceedings of the 48th International Conference on Parallel Processing*. 2019, pp. 1–10.

[7] Masoumeh Ebrahimi. "Reliable and Adaptive Routing Algorithms for 2D and 3D Networks-on-Chip". In: *Routing Algorithms in Networks-on-Chip*. Ed. by Maurizio Palesi and Masoud Daneshtalab. New York, NY: Springer New York, 2014, pp. 211–237. ISBN: 978-1-4614-8274-1. DOI: 10.1007/978-1-4614-8274-1_9. URL: https://doi.org/10.1007/978-1-4614-8274-1_9.

[8] Benhaoua Kamal et al. "Heuristic for Accelerating Run-Time Task Mapping in NoC-based Heterogeneous MPSoCs". In: *Journal of Digital Information Management* 12 (Oct. 2014), pp. 292–302.

[9] Chawade Mahendra, Mahendra Gaikwad, and Rajendra Patrikar. "Review of XY Routing Algorithm for Network-on-Chip Architecture". In: *International Journal of Computer and Communication Technology* (Oct. 2016). DOI: 10.47893/IJCCT.2016.1384.

[10] Armin Mehran, Ahmad Khadem-Zadeh, and Samira Saeidi. "DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip". In: *Ieice Electronic Express* 5 (July 2008), pp. 464–471. DOI: 10.1587/elex.5.464.

[11] Suraj Paul et al. "Adaptive Task Allocation Scheduling on NoC based Multicore Platforms with Multitasking Processors". In: *ACM Transactions on Embedded Computing Systems* 20 (June 2020). DOI: 10.1145/3408324.

[12] Muhammad Sethi and Rehmat Ullah. "Congestion-Aware Routing Algorithm for NoC Using Data Packets". In: *Wireless Communications and Mobile Computing* 2021 (Aug. 2021). DOI: 10.1155/2021/8588646.

[13] Amit Kumar Singh et al. "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms". In: *Journal of Systems Architecture* 56.7 (2010). Special Issue on HW/SW Co-Design: Systems and Networks on Chip, pp. 242–255. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2010.04.007. URL: https://www.sciencedirect.com/science/article/pii/S1383762110000330.

[14] H. Topcuoglu, S. Hariri, and Min-You Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing". In: *IEEE Transactions on Parallel and Distributed Systems* 13.3 (2002), pp. 260–274. DOI: 10.1109/71.993206.

[15] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing". In: *IEEE transactions on parallel and distributed systems* 13.3 (2002), pp. 260–274.

[16] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing". In: *IEEE transactions on parallel and distributed systems* 13.3 (2002), pp. 260–274.

[17] Yu Yao et al. “A communication-aware and predictive list scheduling algorithm for network-on-chip based heterogeneous muti-processor system-on-chip”. In: *Microelectronics Journal* 121 (2022), p. 105367.