Multicore and Network Topology Codesign for Pareto-Optimal Multinode Architecture

Ophélie Renaud, Karol Desnos, Erwan Raffin, Jean-François Nezan

Multicore and Network Topology Codesign for Pareto-Optimal Multinode Architecture

Ophélie Renaud*, Karol Desnos*, Erwan Raffin†, Jean-François Nezan*
*Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164, 35000 Rennes, France. first.last@insa-rennes.fr
†CEPP - Center for Excellence in Performance Programming, Eviden, 35700 Rennes, France. erwan.raffin@eviden.com

***Abstract*—This paper presents an exploration framework for multinode multicore High-Performance Computing (HPC) systems. The proposed method allows to explore rapidly various network topologies to find the optimized solutions in terms of latency for a given application. The proposed method facilitates the in-depth Design Space Exploration (DSE) to identify Pareto-optimal solutions in HPC systems. Experiments demonstrate that our method accelerates HW-SW co-design search by a factor of 3.7 compared to a semi-exhaustive method.**

I. Introduction

The Square Kilometre Array (SKA), an advanced exascale radio telescope, marks a new era in astronomy. This groundbreaking project imposes significant computational challenges on its core, the Science Data Processor (SDP) pipeline. Responsible for handling data from telescopes at an impressive rate of several terabytes per second, the SDP operates within constraints of limited storage and a strict energy budget of 1 Megawatt for 250 Petaflops [1]. To address the SKA computational challenges, the SDP supercomputer would integrate a multinode HPC system. However, allocating resources for complex algorithms in such systems and finding an ideal architecture combination for such algorithms are both known as an NP-complete problem [2]. Given that these algorithms are developed using datasets smaller than those processed by the final system, it is imperative to assess the "at scale" computational requirements of these algorithms to determine their feasibility. The presented work facilitates the rapid simulation of the "at scale" computational needs of the studied algorithms.

This paper explores the dataflow parallel programming paradigm, which inherently expresses parallelism and streamlines the deployment of applications on a selected architecture [3]. Dataflow graph consists of nodes, or actors, representing computations, and directed arcs representing First In First Out (FIFO) buffers. In this paper, we propose a method to automatically identify a suitable multinode and multicore architecture for a given application. The method simulates various network topologies and crucial metrics like final latency, energy consumption, memory management, and associated system costs for each configuration. Driven by pursuing Pareto-optimal architectures, the simulator accommodates customizable input parameter ranges, known as moldable parameters, including the number of nodes, number of cores, and network topology. Employing smart strategies to narrow down the scope of moldable parameters, the method calculates maximum parallelism and memory requirements to streamline the application and eliminate irrelevant configurations, optimizing simulation efficiency. The proposed exploration method has been integrated into a new HW-SW co-design framework called Simulator of the Science Data Processor (SimSDP). Although initially developed for the SKA use case, the proposed approach can be used for the design/implementation of any signal processing system dedicated to big data applications.

The rest of this paper is organized as follows: Section II presents HPC systems, resource allocation process, and related works. Section III describes the proposed method. Section IV outlines the experimental evaluation of the process for narrowing the hardware DSE scope regarding duration and effectiveness. Finally, Section V concludes this paper.

II. Context & Related Work

A. HPC Systems

HPC systems are advanced computing devices designed for the processing of complex tasks. These systems use parallel processing, allowing multiple tasks to run simultaneously for high performance. This paper specifically discusses Distributed Computing System (DCS) that incorporates HPC capabilities. These systems are characterized by multiple interconnected nodes, where each node may consist of multicore processors or specialized accelerators.

The way nodes are connected in the network is crucial for the overall performance of DCS [4]. Network topologies define the structure of connections and communication pathways between nodes. In the context of the exploration for an optimal architecture for a given application, the topology becomes a key parameter to be considered. To narrow the hardware DSE scope and cover common usage scenarios, we focus on the five most prominent families of network topologies, denoted T and illustrated in Figure 1. To clarify the later iterative process each topology is numbered such as Cluster with Crossbar $T = 1$, Cluster with Shared Backbone $T = 2$, Torus cluster $T = 3$, Fat tree cluster [5] $T = 4$, and dragonfly cluster [6] $T = 5$.

To encapsulate the exploration for an adequate architecture tailored to a given application, we define:

$$\alpha(N, C, T) \implies \rho(L_{\text{final}}, M, E, \hat{C}) \tag{1}$$

This work was supported by DARK-ERA (ANR-20-CE46-0001-01).

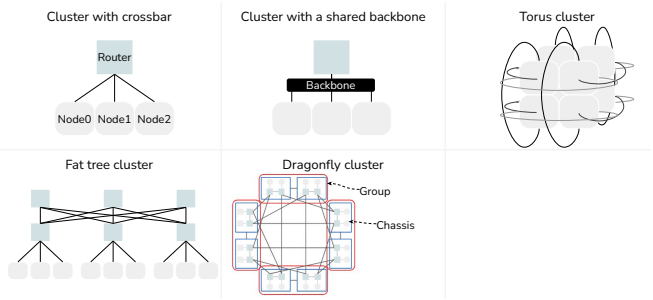

Fig. 1. The five most prominent families of network topologies

Where α represents the architecture parameterized by the number of nodes N, the number of cores per node C , and the network topology T. ρ denotes the Pareto optimal function, evaluating the final latency L_{final}, memory footprint M, energy consumption E, and system cost $\hat{C}$.

B. Resource Allocation for Static Dataflow Model of Computation (MoC)

Dataflow-based resource allocation, and more broadly software synthesis, translates the dataflow graph into executable code, specifically designed for complex computing platforms. This synthesis involves crucial tasks: Scheduling and Mapping (assigning actor execution to Processing Elements (PEs)), Memory Allocation (allocating memory resources near PEs for storing data), and Communication Routing (ensuring data availability and synchronization of computations across the system).

In DCS, efficient resource allocation encompasses a twofold process [7]. First, the actors of the dataflow graph are strategically distributed across multicore nodes, and for each, a software synthesis is performed. Subsequently, another software synthesis is performed to connect nodes to each other.

C. Related work

a) Rapid Prototyping and Simulation Tools: Dataflow-based models and associated prototyping tools have proven their efficiency to accelerate the development of signal-processing application and their deployment on complex architectures. Simulink [8] is commonly used for dynamic system modeling and simulation. However, Simulink lacks specialization for dedicated architectures whether for multi-node HPC or embedded systems. Python libraries offer the capability to parallelize code across multiple nodes; however, a drawback lies in Python's tendency to create full copies and duplicates of all data, leading to inefficiencies in the code. Dask [9] leverages this Python characteristic to parallelize on distributed systems and harnesses dataflow to enhance performance. Nevertheless, scalability issues arise when dealing with large datasets or computationally complex graphs. Parallel and Real-time Embedded Executives Scheduling Method (PREESM) [10] is an open-source rapid prototyping tool, offering robust analysis for heterogeneous multi and many-core single-node targets. SimGrid [11] is an open-source framework specifically designed to model and simulate distributed algorithms on distributed IT architecture. Its purpose is to enable researchers to study distributed algorithms without the need for physical deployment, making it a valuable tool for prototyping and evaluation in diverse distributed environments. While these tools yield relevant results for the resource allocation process and/or simulation, their applicability is limited by the given target architecture.

b) Codesign strategies in HPC: Several solutions exist for determining suitable target architectures for a given application through codesign strategies. One notable tool is ATHENA-SST [12], an extension of the ATHENA analytic tool. This approach allocates resources and refines the architecture exploration based on initial estimations. Subsequently, it leverages the SST tool for detailed simulations. ATHENA-SST primarily specializes in deploying artificial intelligence algorithms on analog and neuromorphic architectures, emphasizing optimization of a specific metric: energy.

c) Pareto Optimal Exploration: Beyond Single-Metric Optimization: The pursuit of Pareto optimal solutions becomes crucial in the context of deploying algorithms on diverse hardware configurations. As highlighted by [13], software parallelism is crucial in optimizing power management for multi-core systems. However, increasing parallelism also raises memory demands. In addition, the relation between final latency and parallelism tends to asymptotically reach the parallel slowdown conforming to Ahmdal's law. In practice, the trend admits several local minimums due to mapping opportunities and task-to-core matching.

D. Previous work

Previous work on SimSDP [7], proposes an efficient, fast, and accurate method to distribute workloads on heterogeneous multinode and multicore architecture leveraging PREESM and SimGrid. The process, illustrated in Figure 2, involves three primary steps: Node-Level Partitioning/Readjustment, Thread-Level Partitioning, and Simulation. The initial step partitions the dataflow graph into subgraphs linked to specific nodes, to achieve a balanced workload distribution. The second step optimizes thread allocation and scheduling within each node. The simulation step covers both intra-node and inter-node executions. The method iteratively readjusts subgraph construction to enhance accuracy by comparing estimated and simulated workload distributions. However, achieving an ideal workload distribution is not the only criterion for achieving high performance. It also depends on accurately sizing the simulated architecture.

To delve into all these aspects we propose here a new method to quickly identify the architecture providing the best final latency for a given application by adjusting the hardware DSE scope and considering the multiple potential local minimums. The proposed method simulates crucial metrics such as final latency, energy consumption, memory usage, and associated costs deploying an application on all architectures of our DSE scope.

III. The Proposed Method

The proposed approach is composed of two main steps. The process starts with an initialization step (III-A) to provide

exploration boundaries for final latency and memory that will be used in the later section (III-B) to guide the user and prevent useless architecture configurations. For each architecture, the method allocates resources thanks to our previous work (II-D) and conducts simulations for four key metrics across five main network topologies (III-C). Figure 2 presents an overview of the method.

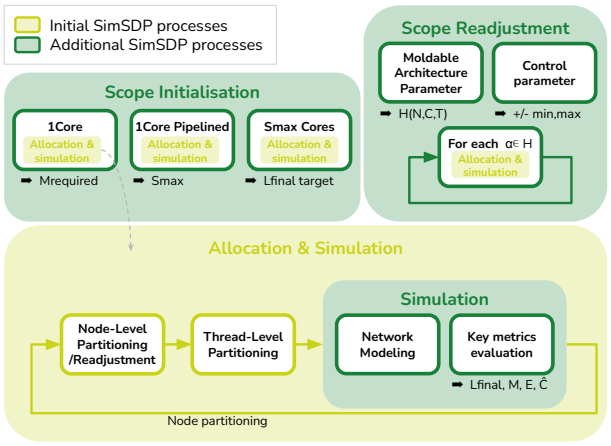

Fig. 2. Visualization of the SimSDP Procedure

A. Scope Initialisation

The initial step aims to evaluate the boundaries of the hardware DSE scope. This step encompasses three first simulations. The first one simulates a single node and single core architecture estimating the minimal memory footprint which later fixes the minimal number of nodes to support the application execution.

The second one simulates the dataflow graph where as many pipeline stages as possible are introduced in the application model giving the maximum theoretical achievable speedup S_{max}. Our method integrates the work of [14] to automatically insert pipelines into a dataflow graph. The maximum achievable speedup is defined as follows:

$$S_{max} = \frac{\text{work length}}{\text{span length}} \tag{2}$$

The work length is defined as the sum of all actor times and the span is the length of the longest sequential path in the dataflow graph. In our case, the span corresponds to a theoretical deployment of the algorithm on an architecture with an infinite number of homogeneous cores without taking into account communication time.

The third one simulates the estimated maximal parallelism such as multiplying the number of nodes by the core count equals the maximum achievable speedup $N \times C = S_{max}$. This step gives an idea of the potential best final latency $L_{\text{final}}(S_{max})$ to verify with the following iterative process.

B. Scope Readjustment

The process proceeds to delineate the hardware DSE scope by establishing parameters for the number of nodes, and number of cores, iterating over this defined range. The *Scope Readjustment* process involves two key phases: the introduction of moldable architecture parameters and the subsequent control of these parameters.

1) Architecture Moldable Parameters: Architecture moldable parameters involve assigning a range of values to a parameter, allowing for the automation of allocating resources and simulating across all delineated architecture configurations feeding the Equation 1. This is achieved through fixing values for the number of nodes ($N_{\text{min}}, N_{\text{max}}$), number of cores ($C_{\text{min}}, C_{\text{max}}$) , and network topology ($T_{\text{min}}, T_{\text{max}}$), each with a defined step size ($\Delta N, \Delta C, \Delta T$). The hardware DSE scope, denoted as H, is mathematically expressed as:

$$\begin{aligned} H = \{ \quad & (N, C, T) \mid \\ & N_{\text{min}} \leq N \leq N_{\text{max}}, \\ & C_{\text{min}} \leq C \leq C_{\text{max}}, \\ & T_{\text{min}} \leq T \leq T_{\text{max}} \} \end{aligned} \tag{3}$$

2) Control Parameter: Moldable parameters are readjusted throughout the process to match the specific needs of the given application and to reduce the exploration time. Given that resource allocation is the most time-consuming step and needs recalculations whenever the number of nodes and cores changes, we focus on adjusting the scope of these two parameters: node number and core count ranges. The network topology range is set by the user without much influence on the search time. Therefore, the parameters to readjust are N_{min}, N_{max}, C_{min} and C_{max}. N_{min} is readjusted considering the application's memory requirements given by the initial step. The minimal node needed to support the entire application is defined as follows:

$$N_{\text{min}} = \lceil \frac{M_{\text{Requirement}}}{N_{\text{Capacity}}} \rceil \tag{4}$$

Where $M_{\text{Requirement}}$is the memory requirement and N_{Capacity} is the node storage capacity.

For each simulated configuration, we determine the maximum level of parallelism achieved when the final latency stops decreasing for at least δ_α consecutive architecture configurations, despite increasing parallelism, and remains less than or equal to the potential best final latency $L_{\text{final}}(S_{\text{max}})$. This ensures that we identify the global minimum of the final latency trend.

$$\begin{aligned} P_{\text{max}} = \max \{ & N \times C \mid \\ & \exists i \geq \delta_\alpha : L_{\text{final}}(\alpha, i) \leq L_{\text{final}}(S_{\text{max}}) \} \end{aligned} \tag{5}$$

Where P_{max} is the maximal parallelism, N the node number and C the cores count, $L_{\text{final}}(\alpha)$ is the final latency obtained on the architecture α, the i^{th} simulated configuration.

C. Simulation

For each configuration defined as $\alpha(N, C, T)$, a model of architecture is automatically generated. The main goal is to analyze how these parameters affect different architectures. The architecture modeling considers these parameters as variables while keeping other parameters at default values that reflect real-world architectures.

1) Architecture Modeling: The architecture modeling includes two phases: multinode modeling and network topology modeling. In the multinode modeling phase, the structure is influenced by the number of nodes (N), and the number of cores (C). This structure impacts resource allocation and is modeled upstream in each iteration. In the network topology modeling phase, the structure is influenced by the number of nodes (N) within the five main topology families. Our focus is solely on comparing these five primary network topologies. Therefore, we propose generic models with basic routing for each, with the only variable being the number of nodes.

- $\text{Crossbar}_{\text{cluster}}$ and $\text{Backbone}_{\text{cluster}}$: These configurations are excluded if the number of nodes exceeds the number of ports on the router R.
- $\text{Torus}_{\text{cluster}}$: Characterized by the parameters "x, y, z " where x, y and z are the dimensions of the torus such as $N = x \times y \times z$. Such torus exists if N is divisible by $x \cdot y \cdot z$.
- $\text{Ftree}_{\text{cluster}}$: Defined by the parameters "$L, \mathbf{D}_{\text{link}}, \mathbf{U}_{\text{link}}, \mathbf{P}_{\text{link}}$" where L is the number of levels, $\mathbf{D}_{\text{link}} = \{N_{\text{pr}}, ..., com\}$ a vector containing the number of downlink for each level, where N_{pr} is the number of nodes or leaf per router and $com = \frac{N}{N_{\text{pr}}}$ is the level of communication. $\mathbf{U}_{\text{link}} = \{1, ..., com\}$ is a vector containing the number of uplink for each level, and $\mathbf{P}_{\text{link}} = \{1, ..., 1\}$ is a vector containing the number of parallel link for each level disabled in our case. The value of L is incremented according to the router capacity and the number of nodes. Such fat trees exists if the number of nodes is a power of two.
- $\text{Dragonfly}_{\text{cluster}}$: Defined by "$G, G_{\text{link}}; C, C_{\text{link}}; R, R_{\text{link}}; N_{\text{pr}}$" Where G is the number of group, G_{link} the number of link between group, C is the number of chassis per group, C_{link} the number of link between chassis, R the number of router per chassis, R_{link} the number of link between routers, N_{pr} is the number of node per router. We compute $N = G \times C \times R \times N_{\text{pr}}$. Such a dragonfly exists if the number of nodes is even.

2) Key metrics: The final latency L_{final} is defined as the elapsed time between the initiation of the first task and the completion of the last task achieved when the application reaches its maximum throughput. It represents the total time taken for the entire dataflow graph to be processed, considering the concurrent execution of tasks on multiple cores.

The memory requirements M are determined by the cumulative size of individual FIFO buffers of the dataflow graph.

The energy consumption E is the sum of node-level energy and link-level energy. The node-level energy is calculated by combining dynamic power multiplied by node execution time and static power multiplied by node voltage. The product of transmission power, link distance, and link bandwidth determines the link-level energy.

The system cost is calculated by summing the individual elements associated with nodes, cores, routers, and links computed when modeling architectures and topologies.

IV. Experiments

A. Experimental Setup

The proposed method is applied to three applications, including an Radio Frequency Interference (RFI) filter from the SKA project, along with two other image processing applications: Sobel and Squeezenet. This diverse set of applications underscores the broad applicability of the proposed method. The resource allocation processes are performed on a desktop computer with an 8-thread Intel i7-8665U processor and 31,2 GB of RAM. Some simulations are subsequently deployed on the multinode cluster *Parasilo* of the grid5000.The proposed method has been implemented into the PREESM rapid prototyping framework in open-source projects.

	Sobel	**RFI**	**Squeezenet**
Final latency	0,52 %	1,06 %	0,98 %
Memory	3,15 %	6,52 %	0,02 %

TABLE I
IN VIVO VS. IN SILICO ERROR ANALYSIS: DEPLOYING SOBEL, RFI FILTER, AND SQUEEZENET ON 5 ARCHITECTURES

In Figure 3, the simulation results for deploying the Sobel dataflow algorithm are presented, considering final latency, memory, energy, and cost. The simulations are conducted across various architectures defined by input moldable parameters. The moldable parameters are define as follow: $\{N_{\text{min}}; \Delta N; N_{\text{max}}\} = \{1; 1; 12\}$, $\{C_{\text{min}}; \Delta C; C_{\text{max}}\} = \{1; 1; 6\}$, $\{T_{\text{min}}; \Delta T; T_{\text{max}}\} = \{1; 1; 5\}$. Subfigure (a) illustrates the results without narrowing the hardware DSE scope, while subfigure (b) includes narrowing the hardware DSE scope. Pareto-optimal architectures are marked with red crosses. Subfigure (c) compares the Sobel application's simulation results for the best node number configuration across the five main topologies concerning the four metrics. The metrics are normalized and inverted to obtain the optimum at 1 in the radar chart. Additionally Table I provides a summary of the average Mean Squared Error (MSE) between the ratio of values obtained on a single core versus the values obtained on the tested architecture. These values are simulated and measured across the five best Pareto optimal architectures retrieved by the method within the scope for the Sobel, RFI filter and Squeezenet application. This indicates the gap between simulation and measurement, and therefore the robustness of our tool. Due to hardware limitations on the Grid5000 cluster, we couldn't precisely measure energy consumption and machine cost for this study. As a result, our focus is primarily on comparing final latency and memory usage, chosen for their critical importance in our research, along with the availability of measurable data.

B. Tune Architecture Simulation Analysis

Figure 3 shows that the method found the best architecture for deploying the Sobel application in less than 15 minutes, which for manual methods can take weeks. Additionally, the results in Figure 3 illustrate that scope tuning narrows down

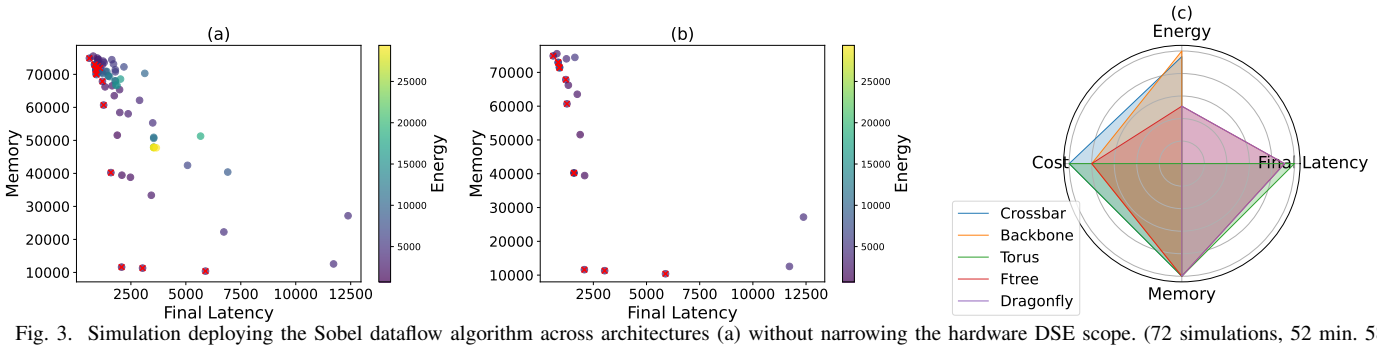

Fig. 3. Simulation deploying the Sobel dataflow algorithm across architectures (a) without narrowing the hardware DSE scope. (72 simulations, 52 min. 58 sec.). (b) narrowing the hardware DSE scope. **(22 simulations, 14 min. 26 sec.)**, and (c) the best node configuration across the 5 main network topologies

the simulation to Pareto-optimal architectures for the Sobel dataflow algorithm deployment, leading to a decreased number of simulation points in Subfigure (b) compared to Subfigure (a). The method divides simulation time by a factor of 3.7, dividing the number of simulations by 3.2 while preserving 59.09% of Pareto-front-safe simulations. The resource allocation process already accelerated thanks to previous work on SimSDP now allow to focus on relevant architectures only. For this Sobel application, optimal final latency results are obtained on an architecture composed of 6 nodes, and 6 cores. Subfigure (c) highlights the impact of network topologies on this 6-node architecture. We obtain the best final latency deploying Sobel on a Torus cluster while Pareto optimal is obtained on a fat-tree cluster more balanced and efficient on several metrics. In the subsequent section, we exploit these findings for comparison with ground truth.

C. In Vivo and Silico comparison

Table I shows that the average MSE between simulated and measured final latency speedup for the three simulated applications on the top five Pareto-optimal architectures is less than 2%. These errors are minimal, reflecting the reliability of the method. The top five Pareto-optimals revolve around a parallelism of : Sobel 36, RFI, and Squeezenet 72, respecting the limit of scope.

V. Conclusion & Future Work

This paper introduces a new dataflow-based co-design method for HPC systems. This method extends the multinode simulator, SimSDP, to rapidly identify architectures optimizing final latency for a given application and evaluate crucial metrics such as final latency, energy consumption, memory usage, and associated cost. The method employs architecture moldable parameters and scope-tuning strategies to narrow down the search for the best combination of the number of nodes, core count, operating frequency, and network topology. The result shows that the method accelerates the search by a factor of 3.7 compared to a semi-exhaustive approach. Future work includes integrating optimization algorithms such as convex problem-solving targeting Pareto optimal solutions and investigating heterogeneous hardware DSE.

References

[1] F. Acero, J.-T. Acquaviva, R. Adam, N. Aghanim, M. Allen, M. Alves, R. Ammanouil, R. Ansari, A. Araudo, E. Armengaud *et al.*, "French ska white book-the french community towards the square kilometre array," *arXiv preprint arXiv:1712.06950*, 2017.

[2] W. F. McColl, *Scalable computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 46–61. [Online]. Available: https://doi.org/10.1007/BFb0015236

[3] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, pp. 24–35, 1987.

[4] D. Xiang, "High-radix interconnection networks," in *New Trends in Computer Technologies and Applications*, S.-Y. Hsieh, L.-J. Hung, R. Klasing, C.-W. Lee, and S.-L. Peng, Eds. Singapore: Springer Nature Singapore, 2022, pp. 3–9.

[5] J. Jacobs, "D-mod-k routing providing non-blocking traffic for shift permutations on real life fat trees," 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:1831393

[6] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray cascade: A scalable hpc system based on a dragonfly network," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–9.

[7] O. Renaud, A. Gougeon, K. Desnos, C. Phillips, J. Tuthill, M. Quinson, and J.-F. Nezan, "SimSDP: Dataflow Application Distribution on Heterogeneous Multi-Node Multi-Core Architectures," in *To be published*. FRANCE: IEEE, 2024.

[8] E. C. Klikpo, J. Khatib, and A. Munier-Kordon, "Modeling multi-periodic simulink systems by synchronous dataflow graphs," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–10.

[9] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th python in science conference*, no. 130-136. Citeseer, 2015.

[10] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming," in *2014 6th european embedded design in education and research conference (EDERC)*, IEEE. FRANCE: IEEE, 2014, pp. 36–40.

[11] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[12] M. Plagge, B. Feinberg, J. McFarland, F. Rothganger, S. Agarwal, A. Awad, C. Hughes, and S. G. Cardwell, "Athena: Enabling codesign for next-generation ai/ml architectures," in *2022 IEEE International Conference on Rebooting Computing (ICRC)*, 2022, pp. 13–23.

[13] S. Holmbacka, E. Nogues, M. Pelcat, S. Lafond, and J. Lilius, "Energy efficiency and performance management of parallel dataflow applications," in *Proceedings of the 2014 Conference on Design and Architectures for Signal and Image Processing*, 2014, pp. 1–8.

[14] A. Honorat, K. Desnos, M. Dardaillon, and J.-F. Nezan, "A fast heuristic to pipeline sdf graphs," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, SAMOS 2020, Samos, Greece, July 5–9, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 139–151.