Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends

Amit Kumar Singh[1], Muhammad Shafique[2], Akash Kumar[1], Jörg Henkel[2]
[1] Department of Electrical and Computer Engineering, National University of Singapore, Singapore
[2] Chair for Embedded Systems (CES), Karlsruhe Institute of Technology, Karlsruhe, Germany
[1]{eleaks,akash}@nus.edu.sg, [2]{muhammad.shafique,henkel}@kit.edu

ABSTRACT

The reliance on multi/many-core systems to satisfy the high performance requirement of complex embedded software applications is increasing. This necessitates the need to realize efficient mapping methodologies for such complex computing platforms. This paper provides an extensive survey and categorization of state-of-the-art mapping methodologies and highlights the emerging trends for multi/many-core systems. The methodologies aim at optimizing system's resource usage, performance, power consumption, temperature distribution and reliability for varying application models. The methodologies perform design-time and run-time optimization for static and dynamic workload scenarios, respectively. These optimizations are necessary to fulfill the end-user demands. Comparison of the methodologies based on their optimization aim has been provided. The trend followed by the methodologies and open research challenges have also been discussed.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time systems and embedded systems

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Multiprocessor Systems-on-Chip, embedded systems, application mapping

1. INTRODUCTION

The maximum operational frequency of a single-core processor has hit the roof due to power dissipation and radio frequency effects. This has forced chip manufacturers to limit the maximum frequency of the processor and shifting towards designing chips with multiple cores operating at lower frequencies [42] [10]. Moreover, the performance demands of modern complex embedded applications have increased substantially which cannot be satisfied by simply increasing the frequency of a single-core processor or by customization of the processor. Instead, there is a need of multiple processors that can cohesively communicate and provide increased parallelism. The underlying concept is to consider applications as conglomeration of many small tasks which can be efficiently distributed on multiple processors in order to execute them in parallel and thereby meeting the increased performance demands [3] [46].

DAC'13 May 29 - June 07 2013, Austin, TX, USA.

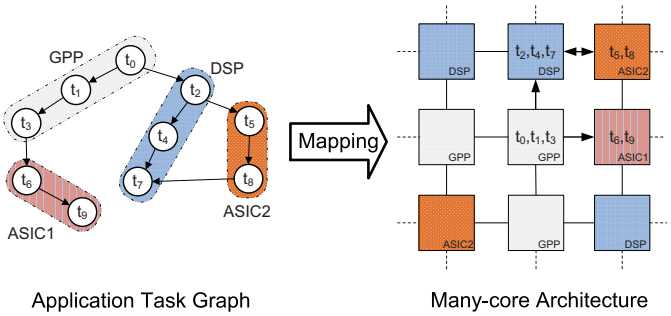


Figure 1: Application Mapping on Many-core System.

With the technological advancement and increasing performance demands, the number of cores in the same chip area has grown exponentially and different types of cores have been integrated. As nanotechnology evolves, it will become feasible to integrate thousands of cores on the same chip [10]. The large number of cores needs to employ Network-on-Chip (NoC) based interconnection infrastructure for efficiency and scalability [33] [7]. The distinct features of different types of cores can be exploited to meet the functional and non-functional requirements. This makes heterogeneous multi/many-core systems (consisting of different types of cores) a formidable computing alternative where applications witness large improvement over their homogeneous (consisting of identical cores) counterpart.

In order to map applications on multi/many-core systems, the applications need to be partitioned (parallelized) into multiple tasks that can be executed concurrently on different cores. An example of partitioned application is shown as *Application Task Graph* in Fig. 1. The application is partitioned into ten tasks (t_0,t_1,...,t_9). The partitioning job can be furnished by state-of-the-art application parallelization tools [14] [53] and manual analysis, which involves finding the tasks, adding synchronization and inter-task communication in the tasks, management of the memory hierarchy communication and checking of the parallelized code to ensure for correct functionality [59]. In case of heterogeneous platforms, a task binding process that specifies the core types on them the task can be mapped along with the cost of mapping is required [84]. The binding process analyses the implementation costs (e.g., performance, power and resource utilization) of each task on different supported core types such as general purpose processor (GPP), digital signal processor (DSP) and coarse grain re-configurable hardware.

Mapping application tasks on multi/many-core system involves assignment and ordering of the tasks and their communications onto the platform resources in view of some optimization criteria such as energy consumption and compute performance. Fig. 1 shows mapping of tasks and their communications on part of a many-core system. The communicating tasks are mapped on the same core or close to each other in order to optimize for the communication delay and energy. The optimization is necessary to satisfy performance constraints of the applications. This necessitates

the need to develop efficient mapping methodologies that take application model, platform model, constraints (e.g., compute performance and power), performance model of inter-process communication (e.g., execution time and energy consumption) and estimate of the worst case execution time (WCET) of the process implementations on different cores (e.g., GPP, DSP, ASIC) as input and provide mappings that satisfy the constraints.

1.1 Mapping Problem and Challenges

Mapping and scheduling problem is similar to *Quadratic Assignment Problem*, a well-known NP-hard problem [28]. Therefore, finding optimal solution satisfying all the given constraints is very difficult and time consuming. Thus, heuristics based on the application domain knowledge need to be employed to find a nearly optimal solution.

Furthermore, the user demands (e.g., performance and power constraints) for each application need to be fulfilled. This necessitates the need to find optimal mapping solutions for each use-case[1] to be supported into the system. The optimal solutions can be explored by advance design-time analysis and then can be used at run-time. However, explosion in the number of use-cases with the increasing number of applications make the analysis unfeasible. For n applications, the analysis needs to be performed for 2^n use-cases. Additionally, such analysis cannot deal with dynamic scenarios such as run-time changing standards and addition of new applications. Run-time management is required to handle such dynamism albeit optimal mapping solutions are not found.

The application mapping problem has been identified as one of the most urgent problem to be solved for implementing embedded systems [57] [60]. This problem is being addressed by several researchers who communicate their views through various forums. A series of dedicated workshops on mapping of applications onto multi-core systems have been started to move beyond state-of-the-art. The mapping methodologies are developed by targeting specific application domain (e.g., multimedia and networking) for the most promising multi-core system.

1.2 Classification of Mapping Methodologies

There could be a number of taxonomies to classify the mapping methodologies, like target architecture based, optimization criteria based, workload based, etc. Broadly, the methodologies can be classified based on workload scenarios and other taxonomies can be included at some hierarchy in the classification as shown in Fig. 2. For static and dynamic workload scenarios, the mapping methodologies perform optimization at **design-time** and **run-time** respectively, which has led them to classify as design-time and run-time methodologies respectively. The methodologies target either **homogeneous** or **heterogeneous** multi-core systems. The run-time mapping requires a platform manager that handles mapping of tasks at run-time. In addition to mapping, the manager is also responsible for task scheduling [51], resource control, configuration control and task migration at run-time. The manager may employ **centralized management**, **distributed management** or mixture of centralized and distributed management. In centralized management, one core of the platform is used as the manager that handles the mapping process. For distributed management, the platform is divided into regions (clusters) and one core in each cluster manages the mapping process inside the cluster. The cluster managers communicate with each other through a global manager to find the best cluster for mapping an application.

Design-time mapping methodologies are suitable for static workload scenarios where a predefined set of applications

[1]Combination of simultaneously active applications.

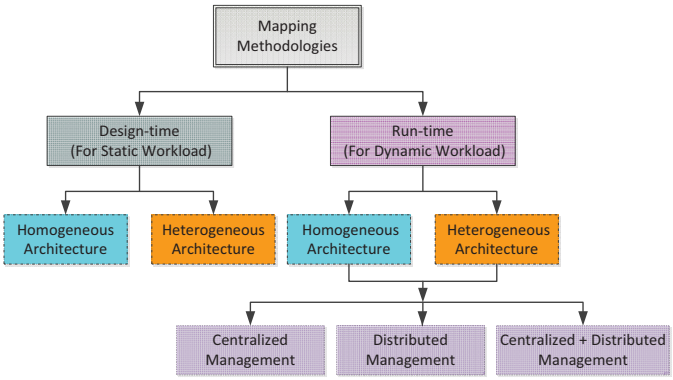


Figure 2: A Taxonomy of Mapping Methodologies.

with known computation and communication behavior and a static platform are considered. They are unable to handle dynamism in applications incurred at run-time (e.g., multimedia and networking applications). Examples of such dynamism could be adding a new application into the system at run-time. Since applications are often added to the platform at run-time (for example, downloading a Java application in a mobile-phone at run-time), workload variation takes place. We witness the need of run-time mapping methodologies to handle such dynamic workloads.

The run-time mapping methodologies face the challenge to map tasks of new applications on the platform resources to satisfy their performance requirements while keeping accurate knowledge of resource occupancy. After mapping tasks, task migration can also be used to revise placement of some of the already executing tasks if the user requirement is changed or a new application has entered into the system.

This paper performs an in-depth *survey* of the mapping methodologies reported in literature based on the earlier mentioned taxonomy. The methodologies have been *analyzed* to highlight their strengths and weaknesses. The trend followed by the methodologies over the *decade* has been observed and reported, which provides an insight for the emerging mapping methodologies. Despite being significant advancement in the development of mapping methodologies, some open issues that need to be addressed in the future are highlighted by analyzing the methodologies reported in the literature.

Paper Organization: Section 2 discusses mapping methodologies that perform design-time optimization while targeting homogeneous or heterogeneous architectures. In Section 3, run-time mapping methodologies are analyzed and elaborated. Section 4 provides the upcoming trends and open research challenges. Finally, we conclude the paper highlighting the important points of mapping methodologies in Section 5.

2. DESIGN-TIME MAPPING

Design-time mapping methodologies have a global view of the system which facilitates in making better decision for using the system resources. As optimization is performed at design-time, the methodologies can use more thorough system information to make decisions. Thus, a better quality of mapping may be achieved as compared to the run-time mapping methodologies that are restricted normally to a local view where only the neighborhood of the task mapping is considered. Most of the mapping methodologies reported in the literature fall under design-time mapping. These methodologies target either homogeneous or heterogeneous architectures.

Table 1 classifies recent works in design-time mapping and shows the target architecture and optimization goal of the mapping methodologies proposed by different authors. The target architecture (*Arch.*) is either homogeneous (*Hom.*)

Table 1: Classification of design-time mapping methodologies.

Author	Arch.	Optimization Goal
Orsila et al. [68]	Hom.	Execution time
Ruggiero et al. [74]	Hom.	Execution time
Satish et al. [76]	Hom.	Execution time
Bonfietti et al. [9]	Hom.	Mapping time & quality
Lin et al. [50]	Hom.	Throughput, Resource utilization,
Murali et al. [65]	Hom.	Energy consumption
Rhee et al. [73]	Hom.	Energy consumption
Chen et al. [16]	Hom.	Energy consumption
Hu et al. [36] [37]	Hom.	Energy consumption, Execution time
Marcon et al. [55] [56]	Hom.	Energy consumption, Execution time
Ascia et al. [5]	Hom.	Energy consumption, Execution time
Meyer et al. [62]	Hom.	Reliability
Thiele et al. [89]	Hom.	Reliability, Temperature
Thiele et al. [88]	Het.	Execution time
Choi et al. [18]	Het.	Execution time
Che et al. [15]	Het.	Execution time
Castrillon et al. [13]	Het.	Execution time
Manolache et al. [54]	Het.	Exploration time, Accuracy
Javaid et al. [41]	Het.	Exploration time, Accuracy
Wu et al. [94]	Het.	Energy consumption
Zhu et al. [100]	Het.	Reliability
Hartman et al. [30]	Het.	Reliability

or heterogeneous (*Het.*). The methodologies aim at optimizing for difference performance metrics in order to fulfill the varying user demands.

Compute Performance: Optimizing for the compute performance is of paramount importance in order to meet the timing deadlines or to minimize the time taken to finish some jobs. The compute performance may refer to total *execution time*, *latency*, *delay*, *period*, *throughput*, *exploration time*, etc., which are related to timing information.

Different well established search approaches have been extensively used to develop design-time mapping methodologies in order to find *optimal* or *near-optimal* placement of tasks on platform cores towards improving the compute performance. For example, Simulated Annealing (SA) is used in [68] [50], Genetic Algorithm (GA) in [18], Tabu Search in [54] and Integer Linear Programming (ILP) in [41]. Orsila et al. [68] optimize execution time and memory consumption by claiming that traditional approaches only focus on the execution time. Lin et al. [50] attempt to find mapping of tasks such that the overall system throughput is maximized. They show an improvement of 20% in the system throughput. Choi et al. [18] propose a GA-based technique for efficiently executing Synchronous Dataflow (SDF) applications on a multi-core system where each core has a limited size of scratchpad memory (SPM). Manolache et al. [54] address the problem of task mapping in the context of multiprocessor applications with stochastic execution times and in the presence of constraints on the percentage of missed deadlines. Javaid et al. [41] propose a methodology consisting of ILP formulation to explore efficient mappings. *These search based approaches provide efficient mapping solutions, but they have high computational costs for large scale problems such as applications with large number of tasks.*

Different pruning strategies have been incorporated to prune the search space, thereby reducing the computational costs. Ruggiero et al. [74] combine Integer Programming with Constraint programming to speed up the executions. They target bus-based architectures that are not scalable and thus the approach enforces scalability issues. Satish et al. [76] propose a decomposition based approach to speed up constraint optimization. They optimize for the schedule length or make-span. Bonfietti et al. [9] propose an approach for throughput-maximal mapping of SDF applications. The approach speeds-up the computation and enhances the efficiency of the search by jump-starting with a high-quality bound and quickly tightening it. Thiele et al. [88] propose a mapping framework called Distributed Operation Layer (DOL), which optimizes for computation and communication time. They integrate an analytic performance analysis strategy into DOL to alleviate the modeling and analysis of systems. Che et al. [15] consider the number of software pipeline stages to map streaming applications on SPM-based embedded multi-core system. The proposed method scales well over a wide range of cores and SPMs. Castrillon et al. [13] propose an algorithm that directly addresses mapping of tasks and their communications. The algorithm is executed repeatedly to compute mappings for real-time applications specified as Kahn Process Network (KPN). *These approaches provide mapping solutions in lesser time than the approaches performing extensive or complete search, but might miss high quality mapping solutions due to pruning of the search space.*

Energy Consumption: Optimizing for the energy consumption of modern embedded systems (e.g., mobile phones, tablets) is important as they are usually operated by standalone power supply like battery. The optimization needs to be performed during the system design and operation in order to increase the operational time.

Murali et al. [65] present a methodology that handles mapping of multiple use-cases while satisfying their performance constraints. The methodology shows a power savings of 54%. Rhee et al. [73] propose an ILP based approach that optimally maps cores onto mesh architecture in order to minimize energy consumption or NoC congestion. The approach achieves 81% energy savings for random benchmarks. Chen et al. [16] propose a multi-step mapping methodology where optimization is performed in different steps of the mapping process. Wu et al. [94] introduce a GA based approach that use *Dynamic Voltage Scaling (DVS)* to reduce the energy consumption by up to 51%. These methodologies show significant energy savings.

Some methodologies that perform optimization for both energy consumption and compute performance are introduced in [37], [56] and [5]. Hu et al. [37] propose a mapping methodology that reduces power consumption by decreasing the energy consumption in communication while guaranteeing the required performance. They methodology provides an energy savings of 51%. Marcon et al. [56] extend the work in [37] and propose a technique called *Communication Dependence and Computation Model (CDCM)*. In addition to communication volume as considered in [37], timing of the communication has been considered. Execution time is reduced by 98% while achieving a significant amount of energy savings. Ascia et al. [5] present a GA based approach that explore Pareto mappings efficiently and accurately while optimizing for performance and energy consumption. *The aforementioned methodologies optimize for compute performance and energy consumption, but do not take reliability of the system into account during that optimization. Therefore, the provided mapping solutions might lead to reduced lifetime of the system.*

Reliability: Design-time mapping methodologies targeting lifetime improvement of multi-core systems are proposed in [62], [89], [100] and [30]. Meyer et al. [62] propose an approach to effectively and efficiently allocate execution and storage slack in order to jointly optimize system lifetime and cost. Thiele et al. [89] propose a thermal-aware system analysis method that produces mappings with lower peak temperature of the system, leading to reliable system design. Zhu et al. [100] exploit redundancy and temperature-aware design planning to produce reliable and compact multi-core systems. Hartman et al. [30] propose a lifetime-aware task mapping methodology that produces mappings with higher lifetimes. These methodologies take preventive measures by performing reliability-aware mapping in order to reduce occurrence of faults in the systems. Such preventive measures

increase lifetime of systems.

2.1 Issues and Limitations of Design-time Methodologies

Most of the design-time methodologies adopt search based approaches (e.g., GA, ILP, SA) that incur high computational costs. Thus, the evaluation time might not be acceptable for large scale problems. However, they provide efficient mapping solutions for small scale systems within acceptable time. The evaluation time can be reduced by efficient pruning of the search space, but at the risk of missing the high quality mapping solutions. The reliability-aware design-time methodologies increase the system lifetime but they cannot overcome the faults incurred in the system.

Further, as the design-time methodologies find placement of tasks at design-time, they are not suitable for run-time varying workloads in the systems and run-time changing environments. Such dynamic workload scenarios require remapping/run-time mapping of applications. Even if these mapping methodologies are inadequate for the dynamic workload scenarios, they might be useful to find the initial task placement, or be optimized to be working at run-time.

3. RUN-TIME MAPPING

In contrast to the design-time mapping, run-time mapping needs to account for the time taken to map each task as it contributes to overall application execution time. Furthermore, the tasks are mapped one by one, unlike the static case where all the tasks are mapped at once by looking globally at the system. Therefore, typically greedy heuristic algorithms are used for efficient run-time mapping in order to optimize performance metrics such as energy consumption, communication latency, execution time, etc. The run-time mapping has several requirements, advantages and issues & research challenges for different available mapping alternatives.

Requirement: The run-time mapping caters for dynamically workload scenarios where mapping of one or more already running applications may need to be reconsidered in case of following requirements:

- Insertion of a new application into the system, which needs resources from the already executing applications.
- Modifying parameters of a running application.
- Killing a running application in order to free it's occupied resources.
- Changing performance requirements of a running application. This might need extra resources for performing extra functionality.
- When current mapping is not sufficiently optimal, it requires (re-)mapping.

Advantages: In addition to the suitability for dynamic workload scenarios, run-time mapping also offers a number of advantages. Some of them are as follows:

- *Adaptability to the available resources*: The available resources vary over time as the applications of the dynamic workload scenario enter at run-time.
- *Ability to enable unforeseeable upgrades*: It is possible to upgrade the system for new applications or changing standards that are not known at design-time, even after the delivery of the system to the end-user.
- *Ability to avoid defective parts of a SoC*: If one or more processing cores are not working properly after production of a SoC, then the defective cores can be disabled before the mapping process. Aging can lead to defective cores that are unforeseeable at design-time.

Mapping Alternatives: At run-time, mapping of new applications to be supported onto a platform can be handled either by performing all the processing at the same time, i.e. on-the-fly processing or by using previously analyzed (DSE) results as shown in Fig. 3. The run-time platform manager handles the mapping of applications by taking the updated resources' status (*Current System Status*) into account.

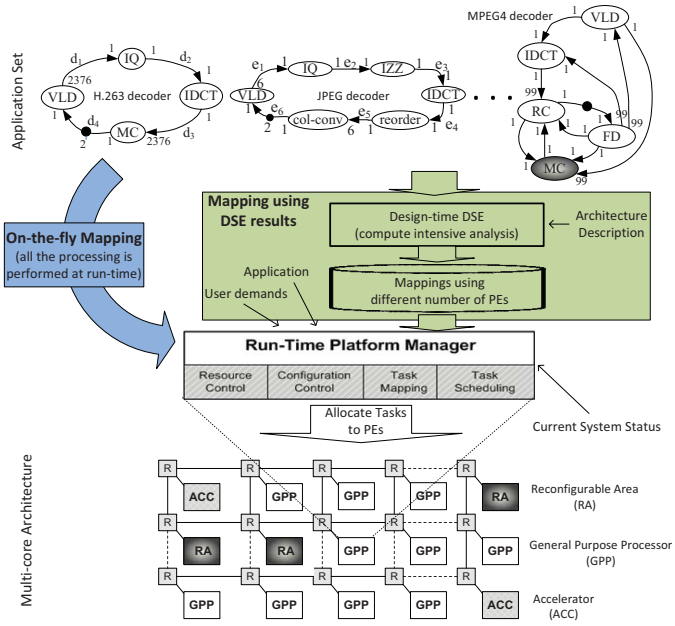


Figure 3: On-the-fly and Hybrid Mapping Flow [81].

For *on-the-fly mapping*, efficient heuristics are required to assign new arriving tasks on the platform resources. These heuristics cannot guarantee for schedulability, i.e., for strict timing deadlines due to lack of any prior analysis and limited compute power at run-time. However, these heuristics are platform independent since they do not use any platform specific analysis results computed in advance. Such heuristics lend well to map unknown applications (not available at design-time) on any platform.

For *mapping using previously analyzed (DSE) results*, the applications to be supported on a platform should be known at design-time. In such cases, light-weight heuristics are required to select the most efficient mappings for each application from the design-time (offline) analyzed mappings stored on the system (*Mappings using different number of PEs*). The selection is done subject to available system resources (extracted from *Current System Status*) and desired performance (*User demands*). The selected mapping is used to configure the platform. Compute intensive analysis is performed at design-time (*Design-time DSE*), facilitating for light-weight run-time platform manager that can configure the applications efficiently. In DSE, *application* and *architecture* description are taken as input and a number of *mappings* are produced. Such mapping methodologies have been referred to as *hybrid mapping* as they take the advantages of both design-time and run-time. The hybrid approach maps applications more efficiently than on-the-fly heuristics. However, flexibility in these approaches is limited, since all potential applications must be known in entirety at design-time and analysis results will be applicable only to the analyzed platform. Therefore, design-time analysis needs to be repeated when the application set or platform changes. Further, storing analysis results introduces additional memory overhead.

3.1 On-the-fly Mapping

Recent works on on-the-fly mapping are classified according to the proposed taxonomy and are listed in Table 2. The table reveals the target architecture (Arch.), control mechanism and optimization goal of the mapping methodologies. The methodologies target homogeneous (Hom.) or

Table 2: Classification of on-the-fly mapping methodologies

Author	Arch.	Control Manager	Optimization Goal
Hong et al. [35]	Hom.	Centr.	Execution time
Shojaei et al. [80]	Hom.	Centr.	Execution time, Solution quality
Moreira et al. [63]	Hom.	Centr.	Execution time, Resource utilization
Chou et al. [19]	Hom.	Centr.	Energy consumption, Communication cost
Mehran et al. [61]	Hom.	Centr.	Energy consumption, Mapping time
Briao et al. [11]	Hom.	Centr.	Energy consumption, Execution time
Qi et al. [72]	Hom.	Centr.	Reliability, Energy consumption
Chou et al. [20]	Hom.	Centr.	Reliability, Energy consumption, Throughput
Coskun et al. [23, 24]	Hom.	Centr.	Reliability, Temperature
Peter et al. [70]	Hom.	Distr.	Execution Time
Theocharides et al. [87]	Het.	Centr.	Execution time
Feng et al. [92]	Het.	Centr.	Execution time
Ahmed et al. [1]	Het.	Centr.	Execution time
Huang et al. [38]	Het.	Centr.	Execution time, Resource utilization
Liang et al. [17]	Het.	Centr.	Execution time, Resource utilization
Nollet et al. [67]	Het.	Centr.	Mapping time & quality
Carvalho et al. [12]	Het.	Centr.	Communication overhead
Smit et al. [84]	Het.	Centr.	Energy consumption, QoS for applications
Braak et al. [86]	Het.	Centr.	Energy consumption, Execution time
Schranzhofer et al. [78]	Het.	Centr.	Energy consumption, Execution time
Singh et al. [83]	Het.	Centr.	Energy consumption, Communication overhead
Hartman et al. [31]	Het.	Centr.	Reliability
Faruque et al. [3]	Het.	Distr.	Execution time, Mapping time, Traffic
Kobbe et al. [46]	Het.	Distr.	Execution time, Traffic
Ebi et al. [27]	Het.	HDistr.	Execution time, Temperature

heterogeneous (Het.) multi-core systems depending upon the requirement of applications. For controlling the system, a *centralized (Centr.)*, distributed (Distr.) or mix of centralized and distributed, i.e. hierarchical distributed (HDistr.) resource management strategy is used to allocate tasks on the resources at run-time. The methodologies aim at optimizing for difference performance metrics.

Compute Performance: The compute performance optimization relates to the timing optimization such as overall execution time and mapping time. Hong et al. [35] change the thread-to-processor mapping at run-time based on the workload variation in order to optimize the performance. An improvement of 29% is achieved in the overall execution time. In [80], the presented heuristic offers additional advantage to trade-off execution time versus solution quality. Moreira et al. [63] present a methodology that first assigns tasks to virtual cores (VCs) aiming to minimize total number of VCs and total bandwidth used while meeting the timing constraints. Thereafter, the VCs are mapped to real cores. Theocharides et al. [87] demonstrate a system-level bidding-based task mapping methodology that provides significant performance improvements when compared to a round robin allocation. Feng et al. [92] perform workload variation aware mapping to optimize the system performance. Ahmed et al. [1] perform adaptive resource management to maintain QoS requirement of application. Huang et al. [38] introduce self-adaptability to the run-time task allocation to achieve high system performance. The adaptability is obtained by dynamically adjusting a set of key parameters based on current resource utilization. Liang et al. [17] take the advantage of shared multi-core reconfigurable fabric to optimize the performance. Nollet et al. [67] describe a run-time task assignment heuristic for efficiently mapping the tasks in a multi-core system containing FPGA fabric tiles. With the presence of FPGA fabric tiles, the heuristic is capable of managing a configuration hierarchy and improves the task assignment success rate and quality of solutions. Carvalho et al. [12] present heuristics where tasks are mapped according to the communication requests and the load in the NoC links. Such consideration reduces the communication overhead, leading to reduced execution time.

The above mentioned methodologies to optimize compute performance use *centralized management (CM)* approach. The CM approach for large systems (many-core systems (consist of thousands of cores)) faces the following problems: *1)* single point of failure, *2)* large volume of monitoring-traffic by the CM, *3)* high computational cost to calculate mapping inside CM and *4)* bottleneck around the CM as every core sends its status to the CM after every instance of mapping [3]. Thus, the CM becomes a hot spot. This necessitates the need of distributed management in order to reduce the monitoring traffic and computational effort.

For *distributed mapping*, the entire system is partitioned into multiple clusters. The resources within each cluster are managed by an individual cluster manager (agent) that communicates with a global platform manager in order to efficiently map an application inside the cluster. The distributed mapping methodology is better than the state-of-the-art run-time mapping methodologies using *Centralized Manager (CM)* approach when many-core systems are targeted. Peter et al. [70] present a heuristic algorithm that is distributed over the processor cores, facilitating its applicability to systems of random size. However, as each core can be considered as a cluster, resource management will become difficult due to linear increment in the number of clusters with the system size. Efficient distributed application mapping methodologies targeting large architectures such as 32×32 and 32×64 systems are presented in [3], [46] and [27]. Faruque et al. [3] consider static applications and focuses on communication, whereas Kobbe et al. [46] consider malleable applications. Ebi et al. [27] consider hierarchical distributed management that targets to trade-off the effectiveness of a centralize approach using global knowledge with the scalability of a fully distributed one.

Energy Consumption: Chou et al. [19] propose a methodology that incorporates the user behavior information in the resource allocation process; that allows system to better respond to real-time changes and adapt dynamically to user needs. This consideration saves 60% communication energy when compared to an arbitrary task allocation technique. Mehran et al. [61] present a *Dynamic Spiral Mapping (DSM)* heuristic algorithm for 2-D mesh topology where placement for a task is searched in a spiral path. The task having maximum degree (connections) is placed at the center of the mesh to facilitate closer mapping of communicating tasks, thereby reducing the communication energy. Briao et al. [11] present strategies where the system turns off idle processors and applies *Dynamic Voltage Scaling (DVS)* to processors with slack to save energy. Smit et al. [84] present an algorithm that first maps tasks needing scarce resources and then all other tasks by taking availability of the platform resources into account. The algorithm minimizes the total amount of energy consumption while providing adequate Quality of Service (QoS) for the application. Braak et al. [86] propose a run-time spatial mapping methodology that spans both the task graph and the multi-core system to find optimal mapping of tasks. Schranzhofer et al. [78] propose a polynomial-time multiple-step heuristic consisting of initial solutions followed by task re-mapping algorithms considering power constraints. Singh et al. [83] incorporate energy consumption measures and multiple tasks per core while mapping tasks according to the communication requests.

Most of the energy-aware mapping methodologies also try to optimize compute performance as shown in Table 2. Thus, they try to fulfill timing constraints while optimizing for the energy consumption. Such optimization is necessary for modern embedded systems that need to perform compute intensive operations for a long time within a limited energy budget. However, these methodologies do not take reliability of the system into account while performing different kinds of optimization.

Reliability: Some recent methodologies that perform optimization for the system reliability are introduced in Table 2. These methodologies take necessary measures to optimize reliability or cure the faults after they have been detected in the system. The detection of faults and their cure is of paramount importance for many real-time systems such as safety-critical, automotive, and avionics. Fail-

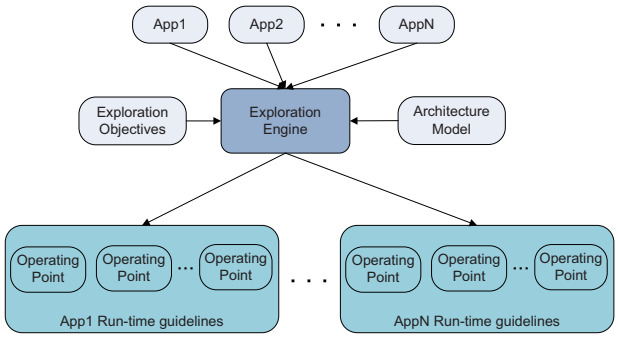


Figure 4: Design-time analysis of applications.

ing to achieve fault-tolerance in these systems may lead to catastrophic consequences. Qi et al. [72] present a technique that optimizes power while considering the system reliability. Chou et al. [20] propose a fault-tolerant application mapping methodology that optimizes system performance and energy consumption, while considering occurrence of different types of faults in the system. Coskun et al. [23, 24] target temperature aware mapping that leads to increased performance and lifetime. Hartman et al. [31] propose a run-time task mapping subsystem that mitigates faults using a wear-based heuristic. The wear-based heuristic is capable of improving system lifetime over temperature-based heuristics. The reliability consideration along with other optimization such as compute performance and energy consumption leads to a better desirable system.

At run-time, some mapping methodologies employ task migration when performance bottleneck is detected or the workload needs to be distributed more homogenously in the whole system [11] [70]. In migration, the tasks are migrated without completely stopping and restarting the already executing applications. Task migration may also be used in case user requirement is changed or a new application has entered into the system in order to revise the placement of some of the already executing tasks. Issues related to the task migration such as the cost to interrupt a given task, saving its context, transmitting all of the data to a new core and restarting the task in the new core are discussed in [66] and [8].

3.2 Based on Design-time Analysis Results

Mapping methodologies based on design-time analysis results perform compute intensive analysis at design-time and use the analyzed results at run-time. Design-time analysis strategies take application and architecture specifications as input and explore mappings with some design objectives (exploration objectives) as shown in Fig. 4. The explored mappings (operating points) provide guidelines for configuring the application at run-time, which is shown as run-time guidelines. The mappings represent trade-offs between different performance metrics. The same analysis strategies can be applied to all the applications (App1, App2, $\cdots$, AppN) one after another, which might need to be supported into the system at run-time, as shown in Fig. 4. *Exploring all the possible mappings for large application and platform size exhaustively is not feasible within a limited time. Therefore, faster analysis strategies having some design objectives are required to explore efficient mappings.*

Single Application Single Mapping Analysis: Most of the design-time analysis techniques reported in literature provide a single mapping for the application. Design-time mapping methodologies reported in Section 2 can be used to find a mapping for an application. Some other such analysis techniques are presented in [64], [2], [45] and [52].They perform exploration in view of some optimization parameters such as computational performance and energy. *The explored single mapping cannot handle dynamism in resource availability and performance requirement at run-time.*

Single Application Multiple Mappings Analysis: Design-time analysis strategies that generate multiple mappings for the application have recently been reported in [58], [98], [4], [85], [29], [93], [43] and [71]. The generated mappings can be used to handle dynamism in resource availability and performance requirement at run-time. In [58] and [98], exploration is performed in view of optimizing for power consumption and performance in order to identify the best performance/power trade-offs. Angiolini et al. [4] optimize for the performance. Stuijk et al. [85] optimize for resource usage. Beltrame et al. [29] optimize for energy and delay. They try to minimize number of simulations required to identify the mappings providing energy/delay trade-offs. Wildermann et al. [93] present multi-objective exploration approach. Jia et al. [43] present an infrastructure called *NASA (Non Ad-hoc Search Algorithm)*, which uses different combination of search strategies to explore the mappings. Piscitelli et al. [71] propose an approach that interleaves the estimations with simulative evaluations in order to ensure that optimal mappings are explored accurately. Most of the analysis strategies use either simulation or an analytical model to evaluate mappings, where simulative evaluations are computationally costly and analytical approaches suffer from accuracy issues. In [43] and [71], simulative and analytical evaluations are combined to perform fast and accurate analysis. These strategies analyze applications one after another. To support the required applications on the system at run-time, they are mapped sequentially by using their analysis results.

Multiple Applications Multiple Mappings Analysis: There has been quite some research in multiple applications DSE. Some researchers focus on scenario based approach where multiple application mapping scenarios are explored at design-time in order to handle dynamism in the number of active applications at run-time [85], [90], [69]. A scenario contains a set of simultaneously active applications referred to as use-case [47] [65] [6]. *The scenario based approaches are not scalable as the number of scenarios increases exponentially with the number of applications, which might become intractable.*

A few strategies that perform mapping using design-time analysis results are presented in [79], [48], [97], [96], [95], [39], [81] and [82]. In [79], analysis result includes only a single mapping having minimum average power consumption. In [48], the authors target to minimize application execution time. In [97] and [96], analysis results include multiple mappings having trade-off in terms of target power consumption and performance. In [95], design-time analysis gives ideal core count and memory required for current state of the application. In [39], a set of process variation-aware schedules is analyzed at design-time. In [81] and [82], analysis results include mappings optimized from throughput point of view for homogeneous and heterogeneous platforms, respectively. The design-time analysis results have been used by run-time platform manager in order to map applications on the platform efficiently. The manager invokes run-time selection strategy to select the best mapping from the design-time analyzed mappings in order to configure the applications on the platform resources.

Reliability-aware Analysis: Reliability (fault) aware design-time analysis can be performed in order to produce solutions that can be used to cure faults incurred at run-time or to reduce the chances of system failure. Some approaches that perform such analysis and use the analysis results at run-time are presented in [25], [49], [77], [26] and [40]. In [25], design-time analysis is performed while aimed at optimizing system's life-time in terms of mean time to failure. In [49], an intensive design-time analysis for all possible failure scenarios is performed. At run-time, tasks are simply

Table 3: Comparison of various approaches for performing design-time analysis and then run-time (RT) mapping

Author	Plat.	Appl.	Maps.	RT	Optimization Goal
Mariani et al. [58]	Fix.	Hom.	Mult.	Yes	Energy consumption, Execution time
Stuijk et al. [85]	Fix.	Hom.	Mult.	No	Resource utilization
Beltrame et al. [29]	Fix.	Hom.	Mult.	No	Energy consumption, delay
Ykman et al. [97]	Fix.	Hom.	Mult.	Yes	Energy consumption, Execution time
Xue et al. [95]	Fix.	Hom.	Mult.	Yes	Resource utilization
Anup et al. [25]	Fix.	Hom.	Mult.	Yes	Reliability
Singh et al. [81]	Gen.	Hom.	Mult.	Yes	Solution quality, Execution time
Yang et al. [96]	Fix.	Het.	Mult.	Yes	Energy consumption, Execution time
Angiolini et al. [4]	Fix.	Het.	Mult.	No	Execution time
Zamora et al. [98]	Fix.	Het.	Mult.	No	Energy consumption, Execution time
Wildermann et al. [93]	Fix.	Het.	Mult.	No	Solution quality, Execution time
Schranzhofer et al. [79]	Fix.	Het.	Sing.	Yes	Energy consumption
Piscitelli et al. [71]	Fix.	Het.	Mult.	No	Solution quality, Execution time
Kwok et al. [48]	Fix.	Het.	Mult.	Yes	Execution time
Huang et al. [39]	Fix.	Het.	Mult.	Yes	Execution time
Lee et al. [49]	Fix.	Het.	Mult.	Yes	Reliability
Schor et al. [77]	Fix.	Het.	Mult.	Yes	Reliability
Derin et al. [26]	Fix.	Het.	Mult.	Yes	Reliability
Huang et al. [40]	Fix.	Het.	Mult.	Yes	Energy consumption, Reliability
Jia et al. [43]	Flex.	Het.	Mult.	No	Solution quality, Execution time
Singh et al. [82]	Gen.	Het.	Mult.	Yes	Energy Consumption, Execution time

remapped using the compile-time decisions. In [77], architectural failures are handled by allocating spare cores during design-time analysis in order to include the evaluation of all the possible failure scenarios. In [26], optimal mappings for all single-fault scenarios in the processing cores are analyzed, which are used by an online task remapping heuristic. The analysis performs optimization to minimize communication traffic and total execution time. In [40], an initial task schedule for different execution modes is generated at design-time. Then, run-time adjustment is performed at regular intervals for optimizing lifetime reliability and energy consumption.

Table 3 shows a comparison of the approaches reported in literature which consider design-time analysis and then analyzed results for run-time mapping. As can be seen, most of the existing approaches perform design-time analysis on fixed (Fix.) or flexible (Flex.) platforms and evaluate mappings that are applicable only to fixed homogeneous (Hom.), fixed heterogeneous (Het.) or a set of heterogeneous (Flex. Het.) platforms. A few approaches consider a *generic (Gen.)* platform and provide multiple mappings that are applicable to large set of platforms. Most of the analysis strategies provide multiple (Mult.) mappings. Some strategies provide support for run-time (RT) mapping that uses design-time analysis results optimized for different requirements.

3.3 Centralized vs. Distributed Management

The run-time mapping methodologies use centralized, distributed or hierarchical distributed resource management techniques. For small platforms such as 4x4 grid of cores, one core can be used as the manager that handles the mapping process. This approach is not scalable with the platform size as the monitoring traffic around the centralized manager increases which may lead to hot-spot resulting in reduced overall performance.

The distributed management caters for the large platforms such as 32x64 grid of cores. The distributed approach reduces the monitoring traffic around the Centralize Manager (CM). However, in a relatively smaller architecture, the CM approach might perform better as the distributed approach incurs additional communication overhead amongst the cluster agents without offering significant advantages. It should be noted that the cluster agents in the distributed approach behave identically to a centralized manager albeit for a small region. A detailed analysis and comparison of the centralized, distributed and hierarchical distributed approaches are provided in the next section.

4. UPCOMING TRENDS AND OPEN CHALLENGES

This section addresses some of the upcoming trends and challenges to be faced to take the mapping methodologies into the next era.

4.1 Hybrid Mapping

The reported mapping methodologies provide three alternatives: design-time mapping, on-the-fly mapping and hybrid (design-time analysis and then run-time use) mapping. Design-time techniques have pre-dominated the reported literature. However, their inability to handle dynamic workload scenarios has led to the formulation of on-the-fly mapping methodologies. On-the-fly strategies surmount the limitation of handling dynamic workloads at run-time but with the fallout of possible non-optimal mapping due to limited compute power at run-time. Recently, the issues of design-time and on-the-fly strategies have been addressed by developing hybrid mapping methodologies that attempt to incorporate the advantages of both. Hybrid strategies combine design space exploration of design-time techniques with the run-time management in order to select mapping configurations that are best suited to newly arriving applications. They involve minimum computation at run-time, facilitating for light-weight run-time manager performing efficient mapping. Our experimental results have shown that run-time mapping gets speeded up by 93% when compared to state-of-the-art on-the-fly mapping methodologies [81]. Although the advantages of hybrid strategy seem promising, it comes with its own trade-offs due to inherent pseudo-dynamic nature and inability to handle new applications without available design-time exploration. With no doubt, hybrid strategies seem to be followed in the field of mapping methodologies but due to their nascent development and lack of in-depth examination, further development of design-time and on-the-fly mapping methodologies will continue hand-in-hand with hybrid strategies.

The hybrid strategies also consider reliability-aware mapping as shown in Table 3. The strategies explore mapping alternatives at design-time for different fault scenarios incurred at run-time. Exploration by considering all the possible fault scenarios while considering different types of faults takes large time that might not be acceptable. This imposes challenge to investigate efficient exploration strategies that should overcome the exploration time bottleneck.

Modern design space exploration (DSE) strategies target optimization for multiple variables in order to satisfy several performance demands. The number of optimization variables is expected to increase with the increasing end user demands. In order to manage the challenges with increased optimization variables, an attention to prune the design space efficiently will be required. Further, heterogeneity of systems is increasing for better fulfilling the demands. This will need to be addressed with care due to the potential explosion in the number of permutations to be considered at each stage of the exploration. The exploration strategies might need to establish an upper limit on the heterogeneity in order to maintain the low complexity.

4.2 Large Scale Architectures

The technological enhancement will enable integration of hundreds and even thousands of cores [10]. Different large scale architectures have already been introduced, like Invasive Computing [32], Angstrom [34], and Intel's TeraFlop [91]. The many-core architectures impose a big challenge to manage their resources at run-time in a scalable manner. To achieve high degree of scalability in many-core architectures, resource management of large number of cores require distributed management as centralized management is not scalable with the number of cores. Some distributed management strategies are introduced in [44], [3], [46] and [27].

Kadin et al. [44] propose a Distributed Dynamic Thermal Management (D^2TM) scheme that delivers about 40% performance improvement over a standard planning scheme

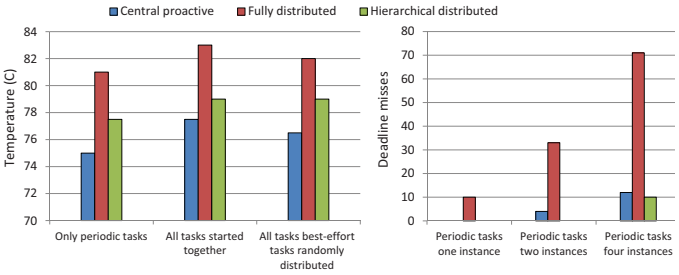


Figure 5: Peak temperatures and missed deadlines in periodic SPEC2006 tasks [27].

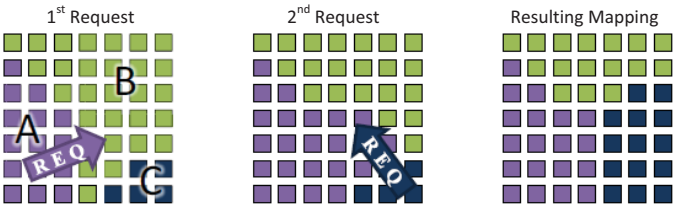


Figure 6: Three applications (A, B, and C) competing for cores in a many-core system [32].

for 16-core system without violating the temperature constraints. For the same system, an optimal central scheme delivers about 42% performance improvement. Fig. 5 shows a comparison of state-of-the-art proactive global centralized (used in [24]), fully distributed (used in [3]) and hierarchical distributed (used in [27]) approaches for peak temperature and deadline misses for applications of SPEC2006 benchmark suite. The temperature and deadline miss values for different simulations (mentioned on horizontal axis) are quoted from [27]. It can be observed that centralized proactive approach provides lower peak temperatures and hierarchical distributed approach shows minimum deadline misses. The lower peak temperatures by the centralized approaches are expected as they have a better view of the system and thus more optimization potential. However, they are limited by their scalability.

A many-core architecture to support invasive computing is introduced in [32]. The invasive many-core architecture contains large number of cores and uses distributed resource management approach [46] to achieve the required degree of scalability. In such large systems, typically many applications execute simultaneously [21] and compete for the available resources. The agent of each application executing in the system try to increase the speedup of its application by acquiring additional cores. Therefore, at run-time, each agent sends requests for cores to the nearby regions. The transfer of cores takes place if gain in the speedup is substantial over the loss for the giving application. Fig. 6 shows an example of three applications A, B, and C competing for resources. In the 1^{st} request, application A requests additional cores and some cores are taken away from application B. In the 2^{nd} request, application C requests more cores and the agents decide to take away some core from applications A and B. The initial mapping gets changed to a more balanced share of resources after the two optimization requests. Additionally, the mappings of applications are optimized over time as new resources might become available due to finishing of tasks of another application.

Fig. 7 shows a comparison of distributed [46] and centralized [75] resource management approaches for various system sizes. The shown results are from CES, KIT, Germany. The centralized scheme in [75] produces competitive and near-optimal schedules. Fig. 7.(a) shows average application speedup, which is computed as the total workload of all applications divided by the sum of the turnaround times of all applications. The results show that DistRM scheme performs better for larger number of cores and achieves about 84% of the mapping quality of the centralized scheme. The centralized scheme always aims at the globally best solution, whereas DistRM scheme performs local changes. Fig. 7.(b) compares accumulated computational effort. In centralize scheme, all computations are performed in a single core and the computational effort increases with the system size. However, in the DistRM scheme, the effort is distributed over the system on different agents and stays constantly low. Fig. 7.(c) plots network utilization, which is calculated by multiplying the amount of messages, the average message size, and the average communication distance. The resulting communication volume required by the DistRM scheme is less for large system sizes. The messages in DistRM scheme are scattered over the NoC, which avoids communication bottlenecks that might encounter in centralized scheme. It can be observed that DistRM scheme requires lower communication volume over the centralize scheme for higher number of applications on large systems such as 32×32. Thus, DistRM scheme is more beneficial for more concurrent applications or larger systems.

Efficient exploitation of the abundant processing power of the available cores is challenging. This is one of the important problem and still needs to be investigated. The investigations need to address how applications from different domain can efficiently utilize large number of processor cores to jointly optimize compute performance, energy consumption and temperature for many-core systems.

4.3 3D Integration of Cores

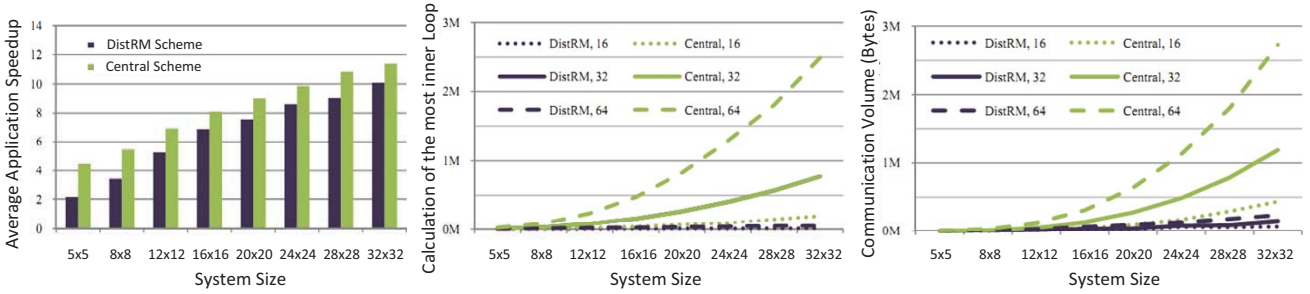
Integration of multiple layers of processor cores into a single device leads to reduced area, power and signal transmission delay. These advantages make 3D multi-core architectures a potential alternative to be used in future high performance computing systems. Despite having several advantages, the 3D high integration density brings major concern in the temperature increase that causes thermal hot spots and high temperature gradients. This might lead to an unreliable system and degraded performance. Efficient thermal management of 3D architectures is challenging and requires investigation of efficient methodologies.

Thermal management techniques for 2D and 3D architectures are reported in [23, 24, 27, 89, 100] and [22, 99], respectively. However, development of efficient mapping methodologies taking thermal issues into account for 3D architectures will continue in foreseeable future. Further, 3D heterogeneous architectures will need to be considered for better fulfilling the increasing functional and non-functional demands. Heterogeneity imposes additional challenges for managing different types of cores.

Some additional challenges also need to be addressed to take the mapping methodologies into the next era. For example, development of efficient programming models for large scale and 3D architectures, efficient synchronization and control of concurrently executing tasks on such architectures and debugging of several concurrent executions if results are not as expected.

5. CONCLUSION

This paper provides a survey of the mapping methodologies targeting multi-core systems. In order to fully utilize the capabilities of multiple cores, the mapping methodologies are inevitably required to efficiently map complex applications onto them. The methodologies reported in the literature target homogeneous or heterogeneous architectures. Heterogeneous architectures provide better performance by exploiting the distinct features of the different type of cores but they are difficult to program as compared to their homogeneous counterpart. These multi-core systems employ NoC-based interconnection infrastructure for efficiency and scalability. The methodologies are classified as design-time



(a) Average application speedup for various system sizes

(b) Accumulated computational effort of both schemes for a workload consisting of 16, 32, and 64 applications on various system sizes

(c) The network utilization of both the schemes on a 2D mesh network for 16, 32, and 64 applications for various system sizes

Figure 7: Comparison of Distributed Resource Management (DistRM) [46] and Centralized [75] scheme.

or run-time methodologies. Their advantages and disadvantages for different type of workload scenarios are described. For dynamic workload scenarios, run-time techniques are proven to be more prevalent and useful. Additionally, they offer several other advantages over design-time techniques such as ability to enable unforeseeable upgrades, ability to avoid defective parts of a system, etc. Based on the analysis of the mapping methodologies, upcoming trends and open challenges are addressed.

6. ACKNOWLEDGMENTS

We also wish to mention that this work is partly supported by Singapore Ministry of Education Academic Research Fund Tier 1 under grant No. R-263-000-655-133 and German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89); http://invasic.de.

7. REFERENCES

[1] W. Ahmed, M. Shafique, L. Bauer, and J. Henkel. Adaptive resource management for simultaneous multitasking in mixed-grained reconfigurable multi-core processors. In *CODES+ISSS*, pages 365–374, 2011.

[2] Y. Ahn, K. Han, G. Lee, H. Song, J. Yoo, K. Choi, and X. Feng. SoCDAL: System-on-chip design AcceLerator. *ACM Trans. Des. Autom. Electron. Syst.*, pages 1–38, 2008.

[3] M. A. Al Faruque, R. Krist, and J. Henkel. ADAM: run-time agent-based distributed application mapping for on-chip communication. In *DAC*, pages 760–765, 2008.

[4] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini. An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration. In *DATE*, pages 1 –6, 2006.

[5] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based noc architectures. In *CODES+ISSS*, pages 182–187, 2004.

[6] L. Benini, D. Bertozzi, and M. Milano. Resource Management Policy Handling Multiple Use-Cases in MPSoC Platforms Using Constraint Programming. In *ICLP*, pages 470–484, 2008.

[7] L. Benini and G. De Micheli. Networks on chips: a new SoC paradigm. *Computer*, (1):70–78, 2002.

[8] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali. Supporting task migration in multi-processor systems-on-chip: a feasibility study. In *DATE*, pages 15–20, 2006.

[9] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *DATE*, pages 897–902, 2010.

[10] S. Borkar. Thousand core chips: a technology perspective. In *DAC*, pages 746–749, 2007.

[11] E. W. Briáo, D. Barcelos, and F. R. Wagner. Dynamic task allocation strategies in MPSoC for soft real-time applications. In *DATE*, pages 1386–1389, 2008.

[12] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes. Dynamic task mapping for mpsocs. *IEEE Des. Test*, pages 26–35, 2010.

[13] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid. Communication-aware mapping of kpn applications onto heterogeneous mpsocs. In *DAC*, pages 1266–1271, 2012.

[14] J. Ceng et al. MAPS: an integrated framework for MPSoC application parallelization. In *DAC*, pages 754–759, 2008.

[15] W. Che and K. S. Chatha. Unrolling and retiming of stream applications onto embedded multicore processors. In *DAC*, pages 1272–1277, 2012.

[16] G. Chen, F. Li, S. Son, and M. Kandemir. Application mapping for chip multiprocessors. In *DAC*, pages 620–625, 2008.

[17] L. Chen, T. Marconi, and T. Mitra. Online scheduling for multi-core shared reconfigurable fabric. In *DATE*, pages 582 –585, 2012.

[18] J. Choi, H. Oh, S. Kim, and S. Ha. Executing synchronous dataflow graphs on a spm-based multicore architecture. In *DAC*, pages 664–671, 2012.

[19] C.-L. Chou and R. Marculescu. User-aware dynamic task allocation in networks-on-chip. In *DATE*, pages 1232–1237, 2008.

[20] C.-L. Chou and R. Marculescu. Farm: Fault-aware resource management in noc-based multiprocessor platforms. In *DATE*, pages 1 –6, 2011.

[21] C.-L. Chou, U. Y. Ogras, and R. Marculescu. Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, pages 1866–1879, Oct. 2008.

[22] A. K. Coskun, J. L. Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici. Dynamic thermal management in 3d multicore architectures. In *DATE*, pages 1410–1415, 2009.

[23] A. K. Coskun, T. S. Rosing, and K. C. Gross. Temperature management in multiprocessor socs using online learning. In *DAC*, pages 890–893, 2008.

[24] A. K. Coskun, T. v. Rosing, and K. C. Gross. Utilizing predictors for efficient thermal management in multiprocessor socs. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, pages 1503–1516, 2009.

[25] A. Das, A. Kumar, and B. Veeravalli. Reliability-Driven Task Mapping for Lifetime Extension of Networks-on-Chip Based Multiprocessor Systems. In *DATE*, 2013.

[26] O. Derin, D. Kabakci, and L. Fiorin. Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors. In *NOCS*, pages 129 –136, 2011.

[27] T. Ebi, D. Kramer, W. Karl, and J. Henkel. Economic learning for thermal-aware power budgeting in many-core architectures. In *CODES+ISSS*, pages 189–196, 2011.

[28] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. 1979.

[29] B. Giovanni, L. Fossati, and D. Sciuto. Decision-theoretic design space exploration of multiprocessor platforms. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, pages 1083–1095, 2010.

[30] A. Hartman, D. Thomas, and B. Meyer. A case for lifetime-aware task mapping in embedded chip multiprocessors. In *CODES+ISSS*, pages 145 –154, 2010.

[31] A. S. Hartman and D. E. Thomas. Lifetime improvement through runtime wear-based task mapping. In *CODES+ISSS*, pages 13–22, 2012.

[32] J. Henkel et al. Invasive manycore architectures. In *ASP-DAC*, pages 193 –200, 2012.

[33] J. Henkel, W. Wolf, and S. Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. In *VLSID*, pages 845 – 851, 2004.

[34] H. Hoffmann et al. Self-aware computing in the angstrom processor. In *DAC*, pages 259–264, 2012.

[35] S. Hong, S. H. K. Narayanan, M. Kandemir, and O. Özturk. Process variation aware thread mapping for chip multiprocessors. In *DATE*, pages 821–826, 2009.

[36] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *ASP-DAC*, pages 233–239, 2003.

[37] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Trans. Comp.-Aided Des. Integ. Cir. Sys.*, (4):551–562, 2005.

[38] J. Huang, A. Raabe, C. Buckl, and A. Knoll. A workflow for runtime adaptive task allocation on heterogeneous MPSoCs. In *DATE*, pages 1 –6, 2011.

[39] L. Huang and Q. Xu. Performance yield-driven task allocation and scheduling for MPSoCs under process variation. In *DAC*, pages 326 –331, 2010.

[40] L. Huang, R. Ye, and Q. Xu. Customer-aware task allocation and scheduling for multi-mode MPSoCs. In *DAC*, pages 387 –392, 2011.

[41] H. Javaid and S. Parameswaran. A design flow for application specific heterogeneous pipelined multiprocessor systems. In *DAC*, pages 250–253, 2009.

[42] A. Jerraya, H. Tenhunen, and W. Wolf. Guest Editors' Introduction: Multiprocessor Systems-on-Chips. *Computer*, (7):36–40, 2005.

[43] Z. J. Jia et al. NASA: A generic infrastructure for system-level MP-SoC design space exploration. In *ESTIMedia*, pages 41 –50, 2010.

[44] M. Kadin, S. Reda, and A. Uht. Central vs. distributed dynamic thermal management for multi-core processors: which one is better? In *GLSVLSI*, pages 137–140, 2009.

[45] J. Keinert et al. SystemCoDesigner Ůan automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Trans. Des. Autom. Electron. Syst.*, pages 1–23, 2009.

[46] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel. Distrm: distributed resource management for on-chip many-core systems. In *CODES+ISSS*, pages 119–128, 2011.

[47] A. Kumar et al. Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. *ACM Trans. Des. Autom. Electron. Syst.*, pages 1–27, 2008.

[48] Y.-K. Kwok et al. A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.*, 66(1):77–98, 2006.

[49] C. Lee, H. Kim, H.-w. Park, S. Kim, H. Oh, and S. Ha. A task remapping technique for reliable multi-core embedded systems. In *CODES+ISSS*, pages 307–316, 2010.

[50] L.-Y. Lin et al. Communication-driven task binding for multiprocessor with latency insensitive network-on-chip. In *ASP-DAC*, pages 39–44, 2005.

[51] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, pages 46–61, 1973.

[52] W. Liu and other. Efficient SAT-Based Mapping and Scheduling of Homogeneous Synchronous Dataflow Graphs for Throughput Optimization. In *RTSS*, pages 492–504, 2008.

[53] A. Mallik et al. MNEMEE - An Automated Toolflow for Parallelization and Memory Management in MPSoC Platforms. In *DAC*, 2011.

[54] S. Manolache, P. Eles, and Z. Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Trans. Embed. Comput. Syst.*, (2):19:1–19:35, 2008.

[55] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner. Time and energy efficient mapping of embedded applications onto NoCs. In *ASP-DAC*, pages 33–38, 2005.

[56] C. Marcon, E. Moreno, N. Calazans, and F. Moraes. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *Computers Digital Techniques, IET*, pages 471 –482, 2008.

[57] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. *IEEE TCAD*, (1):3–21, 2009.

[58] G. Mariani et al. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *DATE*, pages 196–201, 2010.

[59] G. Martin. Overview of the mpsoc design challenge. In *DAC*, pages 274 –279, 2006.

[60] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. Mapping of applications to MPSoCs. In *CODES+ISSS*, pages 109–118, 2011.

[61] A. Mehran, A. Khademzadeh, and S. Saeidi. DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip. *IEICE Electronics Express*, (13):464–471, 2008.

[62] B. H. Meyer, A. S. Hartman, and D. E. Thomas. Cost-effective slack allocation for lifetime improvement in noc-based mpsocs. In *DATE*, pages 1596–1601, 2010.

[63] O. Moreira, J. J.-D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *SAC*, pages 1557–1564, 2007.

[64] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *EMSOFT*, pages 57–66, 2007.

[65] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli. A methodology for mapping multiple use-cases onto networks on chips. In *DATE*, pages 118–123, 2006.

[66] V. Nollet et al. Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles. In *DATE*, pages 234–239, 2005.

[67] V. Nollet et al. Run-time management of a MPSoC containing FPGA fabric tiles. *IEEE Trans. Very Large Scale Integr. Syst.*, pages 24–33, 2008.

[68] H. Orsila et al. Automated memory-aware application distribution for Multi-processor System-on-Chips. *J. Syst. Archit.*, (11):795–815, 2007.

[69] G. Palermo, C. Silvano, and V. Zaccaria. Robust optimization of SoC architectures: A multi-scenario approach. In *ESTIMedia*, pages 7 –12, 2008.

[70] Z. Peter et al. A Decentralised Task Mapping Approach for Homogeneous Multiprocessor Network-On-Chips. *International Journal of Reconfigurable Computing*, 2009.

[71] R. Piscitelli and A. Pimentel. Design space pruning through hybrid analysis in system-level design space exploration. In *DATE*, pages 781 –786, 2012.

[72] X. Qi, D. Zhu, and H. Aydin. Global Reliability-Aware Power Management for Multiprocessor Real-Time Systems. In *ERTCSA*, pages 183–192, 2010.

[73] C.-E. Rhee, H.-Y. Jeong, and S. Ha. Many-to-Many Core-Switch Mapping in 2-D Mesh NoC Architectures. In *ICCD*, pages 438–443, 2004.

[74] M. Ruggiero et al. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *DATE*, pages 3–8, 2006.

[75] G. Sabin, M. Lang, and P. Sadayappan. Moldable parallel job scheduling using job efficiency: an iterative approach. In *JSSPP*, pages 94–114, 2007.

[76] N. Satish, K. Ravindran, and K. Keutzer. A decomposition-based constraint optimization approach for statically scheduling task graphs with communication delays to multiprocessors. In *DATE*, pages 57–62, 2007.

[77] L. Schor et al. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *CASES*, pages 71–80, 2012.

[78] A. Schranzhofer, J.-J. Chen, and L. Thiele. Power-Aware Mapping of Probabilistic Applications onto Heterogeneous MPSoC Platforms. In *RTAS*, pages 151–160, 2009.

[79] A. Schranzhofer, J.-J. Chen, and L. Thiele. Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms. *IEEE Transactions on Industrial Informatics*, (4):692 –707, 2010.

[80] H. Shojaei et al. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In *DAC*, pages 917–922, 2009.

[81] A. K. Singh, A. Kumar, and T. Srikanthan. A Hybrid Strategy for Mapping Multiple Throughput-constrained Applications on MPSoCs. In *CASES*, pages 175–184, 2011.

[82] A. K. Singh, A. Kumar, and T. Srikanthan. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. *ACM Trans. Des. Autom. Electron. Syst.*, pages 1–29, 2013.

[83] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *J. Syst. Archit.*, pages 242–255, 2010.

[84] L. Smit et al. Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. In *FPT*, pages 421–424, 2004.

[85] S. Stuijk, M. Geilen, and T. Basten. A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour. In *DSD*, pages 548–555, 2010.

[86] T. D. ter Braak et al. Run-time spatial resource management for real-time applications on heterogeneous MPSoCs. In *DATE*, pages 357–362, 2010.

[87] T. Theocharides et al. Towards embedded runtime system level optimization for MPSoCs: on-chip task allocation. In *GLSVLSI*, pages 121–124, 2009.

[88] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping Applications to Tiled Multiprocessor Embedded Systems. In *ACSD*, pages 29–40, 2007.

[89] L. Thiele, L. Schor, H. Yang, and I. Bacivarov. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *DAC*, pages 268–273, 2011.

[90] P. van Stralen and A. Pimentel. Scenario-based design space exploration of MPSoCs. In *ICCD*, pages 305 –312, 2010.

[91] S. Vangal et al. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In *ISSCC*, pages 98–589, 2007.

[92] F. Wang et al. Variation-aware task and communication mapping for mpsoc architecture. *IEEE TCAD*, (2):295 –307, 2011.

[93] S. Wildermann, F. Reimann, D. Ziener, and J. Teich. Symbolic design space exploration for multi-mode reconfigurable systems. In *CODES+ISSS*, pages 129–138, 2011.

[94] D. Wu, B. M. Al-Hashimi, and P. Eles. Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems. In *DATE*, page 10090, 2003.

[95] L. Xue, O. ozturk, F. Li, M. Kandemir, and I. Kolcu. Dynamic partitioning of processing and memory resources in embedded MPSoC architectures. In *DATE*, pages 690–695, 2006.

[96] P. Yang et al. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *ISSS*, pages 112–119, 2002.

[97] C. Ykman-Couvreur et al. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *IET Comp. Dig. Techn.*, (2):123 –135, 2011.

[98] N. H. Zamora, X. Hu, and R. Marculescu. System-level performance/power analysis for platform-based design of multimedia applications. *ACM Trans. Des. Autom. Electron. Syst.*, pages 2:1–2:29, 2007.

[99] X. Zhou, J. Yang, Y. Xu, Y. Zhang, and J. Zhao. Thermal-aware task scheduling for 3d multicore processors. *IEEE Trans. Parallel Distrib. Syst.*, pages 60–71, 2010.

[100] C. Zhu, Z. P. Gu, R. P. Dick, and L. Shang. Reliable multiprocessor system-on-chip synthesis. In *CODES+ISSS*, pages 239–244, 2007.