Fault Tolerant Task Mapping on Many-Core Arrays

Colin Bonney, Pedro Campos, Nizar Dahir, Gianluca Tempesti

Department of Electronics, University of York
Email: {cab523, pedro.campos, nizar.dahir, gianluca.tempesti}@york.ac.uk

***Abstract*—This paper presents an approach for generating fault tolerant task mappings of applications, represented as an application process graph (APG), to a many-core array. The approach uses a multi-objective evolutionary algorithm (EA) to evolve a range of viable task mappings through the optimization of fault tolerant properties and performance criteria. Fault tolerant properties are chosen to promote task mappings that enable quick, low-cost recovery in response to fault conditions. Performance criteria promote mappings with lower network traffic. Fault tolerant properties and performance criteria tend to promote different arrangements of tasks thereby creating a range of viable mappings, represented as a Pareto front, that can be used as a pool from which a single mapping can be selected based of the prevailing demands of the system. Analysis of the evolved task maps show that they are resilient to possible fault conditions and exhibit graceful degradation of performance.**

***Index Terms*—Many-core, task mapping, fault tolerance, graceful degradation, graceful amelioration, multi-objective, evolutionary algorithm.**

I. Introduction

The many-core system architecture is seen as a solution to increase the performance of processing systems compared to single-core and multi-core processor designs [1]. A single-core processor has a single complex core. Multi-core processors typically have 2 - 16 cores which are still large, feature rich and complex. Many-core devices are based on an array of identical, smaller, less complex cores numbering in the hundreds or thousands. While the performance of each individual core will be lower than that of the cores in complex single-core or multi-core devices, the combined performance of all the cores has the potential to be significantly greater.

In addition to offering increased overall performance, many-core systems also provide an infrastructure suitable for implementing fault tolerance mechanisms. When individual cores fail, a many-core system has the potential for reallocating the work of failed cores to other functioning cores resulting in a graceful degradation of performance, a distinct improvement over the complete processor failure of single-core systems. Even when all cores are functioning the many-core architecture offers the possibility of improving performance through more efficient allocation of resources. This could, for example, mean using more cores at a lower clock rate and hence lower power or distributing the allocation of tasks to implement a dark-silicon strategy to evenly spread heat generation across the device [2].

Section II reviews previous work in the areas of task mapping and fault tolerance schemes in network-on-chip sytems (NoCs) and many-core arrays. Section III discusses the task mapping problem, the graph representations of the application, the physical many-core array, and task maps. Section IV discusses the evolutionary algorithm (EA) and the fitness functions used to direct the search for task mappings. Section V analyses the resilience of the task maps produced by the EA.

II. Related Work

Early research into task mapping in NoC designs use techniques other than EAs to optimize various objectives such as maximising energy efficiency using a branch and bound algorithm [3], and bandwidth constrained mapping using Dijkstras shortest path algorithm [4]. Both of these approaches create design-time static maps, as does the two-step genetic algorithm approach of mapping a task graph to IP nodes in an NoC [5]. In [6] a run-time multi-objective EA was applied to the two objectives of performance and power consumption while using a static x-y routing. Another approach created a database of optimized task mappings at design time, one for each input application, that are further optimised at run time based on the current status of the system to optimise communication energy, system lifetime or both [7].

A hardware based approach to fault tolerance in which the device attempts to present N cores from a pool of $M + N$ cores as long as there are no more than M fault cores is described in [8], however this approach hides the actual core topology from the application which consequently cannot take into account the path lengths between communicating tasks to reduce network traffic.

Software only approaches to fault tolerance include task remapping in the event of core failures to reallocate tasks amongst healthy cores. In [9] multiple tasks are allocated to each core, such that the cost of task migration is balanced with the overall performance of the resultant mapping, while in [10] the placement of spare cores at either the edges of the array or randomly across the array is coupled with a fault aware algorithm with the goal of minimizing the communication energy consumption, while maximizing the overall system performance. In [11] modified particle swarm optimization (PSO) algorithms are used to balance combinations of two of the objectives of reliability of switching elements, power consumption and communication time. Two modified PSO algorithms are used, one including elements of EAs and the other incorporating elements of simulated annealing.

III. Task Mapping

This paper investigates a methodology to identify task mappings of an application process graph (APG) to cores of a many-core array while balancing the fault tolerance properties of the mappings against the performance of the system. The process allocation problem is NP-hard with the complexity increasing as a factorial of the size of the array [12], [13].

We propose the use of a multi-objective evolutionary approach using the NSGA-II [14] non-dominated sorting algorithm with two objectives: performance and fault tolerance as defined in section III-B. NSGA-II has been chosen as it is a well researched algorithm that has successfully been applied to a wide range of problems. The solution is required to be scalable to many-core arrays containing 1000's of cores. In order to keep the problem size manageable our many-core model makes a logical division of the available cores into autonomous regions which will manage the distribution of tasks locally while cooperating with neighbouring regions to achieve global task distribution. The multi-region approach allows the problem of task allocation to be subdivided into a number of smaller tasks of manageable size. Our early research suggests that array sizes in the range of 6×6 to 8×8 are manageable in terms of the processing required to find good quality task mappings using an evolutionary approach. This paper illustrates the results of our investigations of the mapping problem using a single region with an array size of 6×6.

A. Many-Core Model

The key components of this model for the work presented in this paper are the following:

Many-Core Array: The many-core architecture is a 2-D array of homogenous computing nodes with no predetermined functionality. Each *node* consists of a *router* that communicates with its nearest orthogonal neighbours and a *processing core* attached to the router. As mentioned the many-core array will be logically divided into autonomous regions which manage the local distribution of tasks while cooperating with neighbouring regions to achieve global task distribution.

Application Process Graph (APG): A directed acyclic graph (DAG) representing an application broken down into processes represented as nodes and data transfers between processes represented as edges. Each process is identified by a label Pn : $n \in N$ where N is the number of nodes in the graph. The APG in fig 1(a), used in this research, is a randomly selected, representative example of graphs created by a graph generator [15].

Network Topology Map (NTM): The physical many-core array used in this research is an $r \times c$ lattice arrangement of r rows and c columns of processing nodes interconnected by communication channels, fig 1(b). The processing nodes consist of a routing node connected to adjacent routing nodes on the four compass points and a processing core. An NTM is used to model the core and channels of the many-core array and maintain an inventory of the current status of elements of the many-core array. The map consists of $r \times c$ cores and $(r \times c - 1) + (r - 1 \times c)$ bidirectional channels. Cores are modelled by an ordered tuple representing their row and column coordinates and their status (r, c, s), where the tuple $(0, 0, s)$ refers to the top left hand core of the array. The status of the core is either g for good for f for failed. The channels are modelled by a tuple (r_s, c_s, r_t, c_t, s) where r_s and c_s are the coordinates of the source node of the channel and r_t and c_t are the coordinates of the target node of the channel. The status s can take the same values of g and f as used for the cores.

Task Map: The task map gives details of the use of each core in the many-core array and the location of each process of the APG within the physical cores, (fig. 1(c)). Each core is defined by an ordered tuple representing its row and column coordinates and the name of the processes allocated to it (r, c, p) where r is the row coordinate, c is the column coordinate and p the name of the process. The process p can be the id of one of the application process nodes, i if the core is idle, or f if the core is faulty. Fig 1(c) shows the results of mapping the APG in fig 1(a), to a 6x6 square lattice many-core array, where there are no faulty cores.

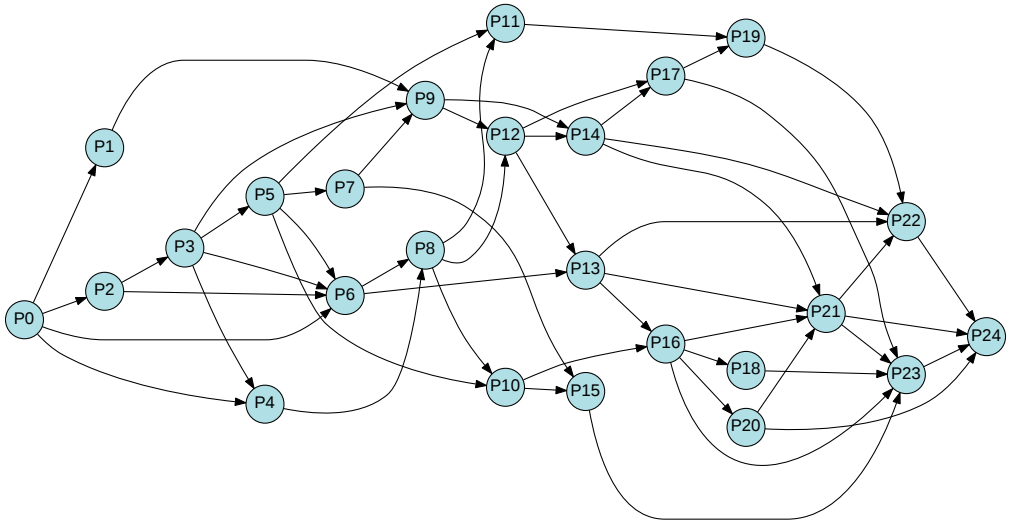
B. Metrics

The work presented in this paper uses the two metrics of *fault tolerance* and *performance* to calculate the fitness values of the two objectives of the multi-objective EA. The fault tolerance and performance cost metrics, discussed below, are orthogonal in that the fault tolerant metric results in graphs where idle cores are distributed evenly across the array, so pushing apart processing tasks, while the performance metric pushes communicating tasks closer together.

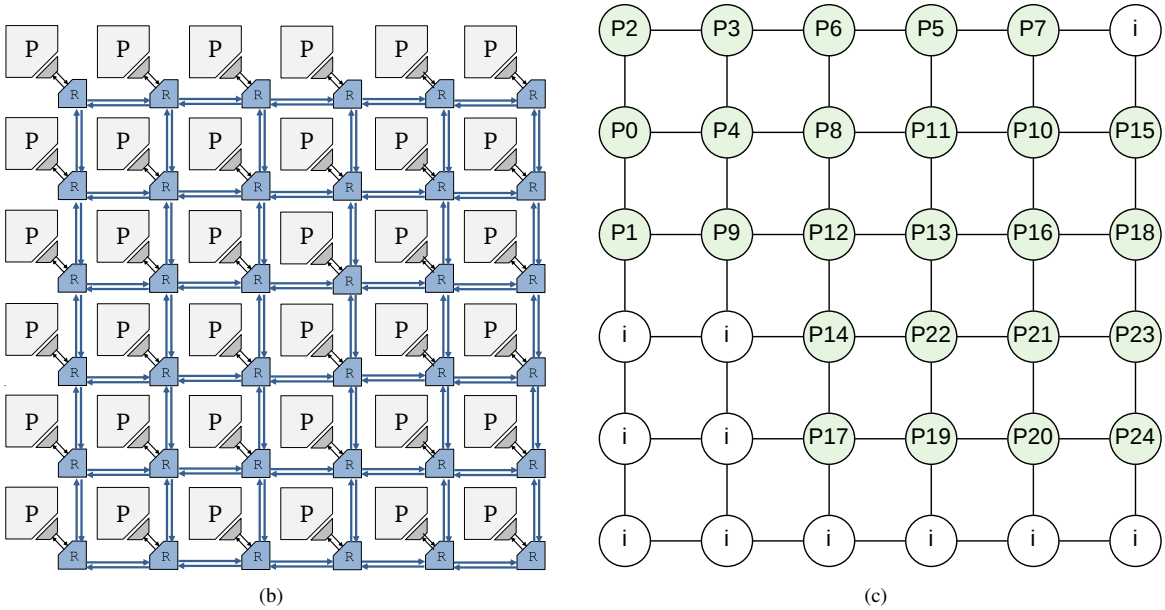
Fault tolerance metric: Fault tolerance is achieved through the placement of *spare* or *idle* cores amongst the cores that are processing tasks.

In the event of a core failure the task executing on the core needs to be migrated to an idle core. The ultimate objective of fault tolerance is to find a mapping that provides minimal disruption in the case of a fault occurring within the array, thereby achieving graceful degradation. Our assumption is that the cost of migration and disruption to processing increases in proportion to the distance between the failed core and the core that the task will be migrated to. This cost takes into account the energy and network traffic required by the migration and the disruption to the task that such a migration would imply. Placing idle cores amongst processing cores to minimise the distance between each processing core and its nearest idle core will correspondingly reduce the average cost of migration of a task to an idle core in the event of core failure.

The fault tolerance metric is thus defined as the sum of the distances between each core running a task and its closest idle core. When an idle core is adjacent to a processing core, i.e. one step away, a fault tolerance cost of zero is assigned to the core. When t steps are required to reach the nearest idle core the fault tolerance cost will be $t - 1$. A fault tolerance cost of



(a)



(b) (c)

Fig. 1. Mapping an APG to a many-core array. (a) the 25-node APG used for this work; (b) the 6x6 many-core array, each node consisting of a processing core (P) and a routing node (R); (c) a possible mapping of the APG onto the many-core array.

zero for the task map as a whole indicates that each processing core is adjacent to at least one idle core. A processing core that has more than one adjacent idle core is not regarded as having any additional benefit from the additional adjacent idle cores. The fault tolerance metric is expressed as:

$$Ftol = \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} distance((r,c), nearestIdle(r,c)) - 1 \quad (1)$$

Where $Ftol$ is the fault tolerance cost of the whole array, r and c are the number of rows and columns in the array, $distance$ is a function that returns the rectilinear distance between the node (r, c) and its closest idle core, and $nearestIdle$ is a function that returns the coordinates of the nearest idle core to the processing core.

Performance Metric: The performance metric used in this paper is a measure of the distance between communicating tasks, which is only one of many possible performance metrics. Communication traffic is one of the most important factors in the performance of a many-core array. Reducing the distance between communicating task pairs also reduces the overall communication traffic across the network resulting

in fewer bottlenecks and lower power consumption. The performance cost metric is therefore measured as the sum of the distances between communicating task pairs. The measure is analogous to the fault tolerance measure in that communicating tasks that are adjacent are given a value of zero and when there are t steps between a pair of communicating tasks a performance value of $t-1$ is given to the communicating pair.

For communicating task pairs that have a least one minimal length path, the performance cost will be the rectilinear distance between the tasks minus 1. Where no minimal length paths are available due to channel faults the cost will be calculated as the (minimum) distance for a non-minimal length path.

If every pair of communicating tasks are adjacent then the performance cost for the mapping will be zero. The performance metric is expressed as:

$$Perf = \sum_{i=0}^{e-1} distance(source_i, target_i) - 1 \qquad (2)$$

Where $Perf$ is the performance cost of the whole array and e is the number edges in the APG, $source_i$ and $target_i$ are the locations of the source and target tasks of the communicating task pair i, in the many core array and $distance$ is the same function used in equation 1.

IV. Multi-Objective Evolution of Task Maps

A. Assumptions and Constraints

Experiments were conducted using the 25 node application graph show in 1(a) mapped to a 6×6 many-core array. Preliminary work determined that the maximum graph size that can be mapped to a 6×6 array such that each processing core is adjacent to an idle core is a graph of 26 nodes and that the problem of finding mappings for graphs close to a size of 26 nodes is an interesting area for research. This has guided our decision to conduct experiments with the combination of a 25, 26 and 27 node graphs on a 6×6 array. Results for different graph sizes and connectivity levels will be analysed in the following section.

All experiments used a population size of 100 individuals, evolved over 1000 generations, evaluating a total of 100,000 individuals, to produce the Pareto front points for analysis.

Seed mappings for the EA were obtained using a selection of simple deterministic algorithms to generate a variety of mappings that outperform randomly generated mappings together with a collection of randomly generated mappings.

We assume that non-minimal path routing algorithms are available, but that routing via minimal length paths is preferred. Task maps that require a non-minimal route will have a higher performance cost due to the additional length of the non-minimal path, providing an evolutionary pressure against such solutions [16], [17].

B. The Pareto Front

The metrics of fault tolerance given in equation 1 and performance given in equation 2 give conflicting pressures to the evolutionary process. A low cost for fault tolerance is achieved when idle cores are evenly distributed across the many-core array and therefore between processing cores, resulting in a pressure pushing processing cores away from each other. The performance metric gives a low cost when communicating cores are adjacent resulting in a pressure that brings cores closer together. The consequence of these conflicting pressures is that the task mapping solutions found by the EA produce a Pareto front of equally valid solutions.

Evolutionary runs were carried out using the seed population as described in section IV-A. An individual is encoded as a task map, and a single mutation is applied to each parent, implemented as a *single pair permutation*, whereby the processes of two nodes are exchanged. This permutation can involve both processing and idle cores. The multi-objective EA explores the solution space in search of solutions which present a good trade-off between performance and fault tolerance. The goal of the multi-objective approach is to *minimise* the fitness value of both the performance and fault tolerance metrics.

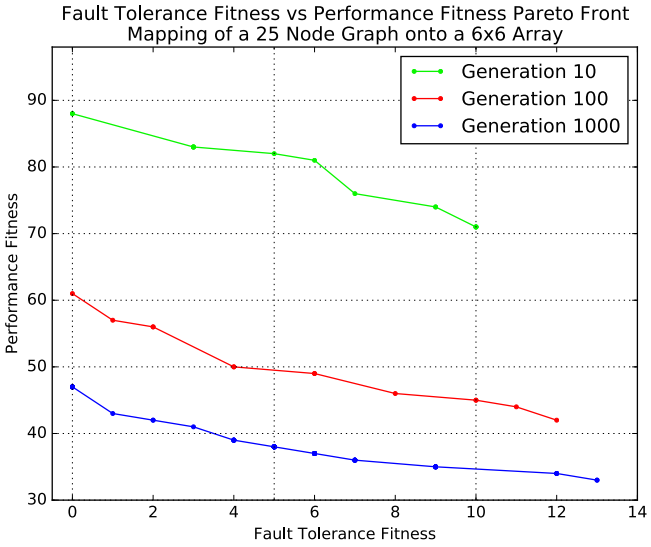


Fig. 2. Pareto fronts for the evolution of task maps for a 25-node graph on a 6x6 array.

For any given graph size on a given many-core array there are many possible solutions which can be represented as a series of two-dimensional Pareto fronts. Fig 2 shows the non-dominated Pareto fronts obtained for generations 10, 100 and 1000 using the NSGA-II algorithm. The points of a non-dominated Pareto front represent a set of equally valid solutions. The points differ only in their suitability when viewed from a specific perspective. Different points may be selected as a suitable solution to fulfil a specific, perceived need. For example, if performance is of paramount importance, then a point can be chosen with a good performance value and poorer fault tolerance value. If fault tolerance is paramount then a point with the good fault tolerance and poorer performance can be chosen. The specific point chosen may be determined

automatically based on the prevailing requirements placed upon the system or on the environmental conditions within which the system is operating.

For illustration purposes, three individuals have been chosen from the Pareto front to highlight the differences between them. Figure 3 shows the task maps corresponding to three different points chosen from the Pareto front, each representing a unique trade-off. The task map of figure 3(a) has good fault tolerance and poor performance, resulting in a distribution of idle cores such that each processing core is adjacent to an idle core. The task map of figure 3(b) has weak fault tolerance and very good performance, which has resulted in the processing cores being close together with the majority of idle cores being pushed to the outside of the array. The task map in figure 3(c) has a more balanced trade-off between the two metrics, resulting in a distribution of idle cores between those in 3(a) and 3(b) leaving only four of the processing cores without an idle core as a neighbour.

V. Fault Resilience

As described in section III-B, the purpose of the fault tolerance metric is to promote task mappings that suffer minimal impact on performance when faults occur. This section focuses on an evaluation of the impact of faults on the performance of the discovered solutions beginning by describing the type of faults injected as well as the run-time mechanisms involved in fault recovery, followed by an evaluation of the impact on performance that these faults can have on different mappings.

A. Classification of Faults

Core Faults are defined as the failure of a processing core. The function of the routing node and channels to adjacent nodes are unaffected. If a core is processing a task when failure occurs, then the task needs to be migrated to an idle core.

Routing node faults are defined as a failure of the routing node with the associated loss of the attached core and channels, equivalent to the simultaneous failure of the processing core and the channels to all adjacent nodes. If the core is processing a task when the routing node fails, then the task needs to be migrated to an idle core.

B. Graceful Degradation

The concept of *graceful degradation* assumes a set of mechanisms that allow an application to continue running on faulty hardware, with degraded performance. Therefore, this concept necessitates an immediate response of the system to a fault. When a core fault occurs, the task being executed on the failed core is migrated to the nearest idle core allowing the system as a whole to continue to function even though performance may be degraded. The failed core and consequent task migration results in a revised NTM and task map. The performance and fault tolerance metrics are likely to be degraded compared to the original mapping since the distance between communicating task pairs and processes and their idle cores are likely to increase. A routing node fault will carry the same consequences, with the additional impact of potentially severing minimal paths as a consequence of communication channels becoming unusable.

C. Engineered Reference Solution

In order to provide a reference mapping against which the evolved multi-objective mappings can be compared, an engineered mapping that only considers the performance metric was generated. The engineered mapping, shown in fig 1(c), was generated by the evolutionary algorithm running in a single objective mode where only the performance metric was calculated, which provided a mapping with a performance metric superior to any of the multi-objective evolved mappings. The engineered solution is the right-most box plot in figures 5, 6 and 7.

D. Reliability Evaluation

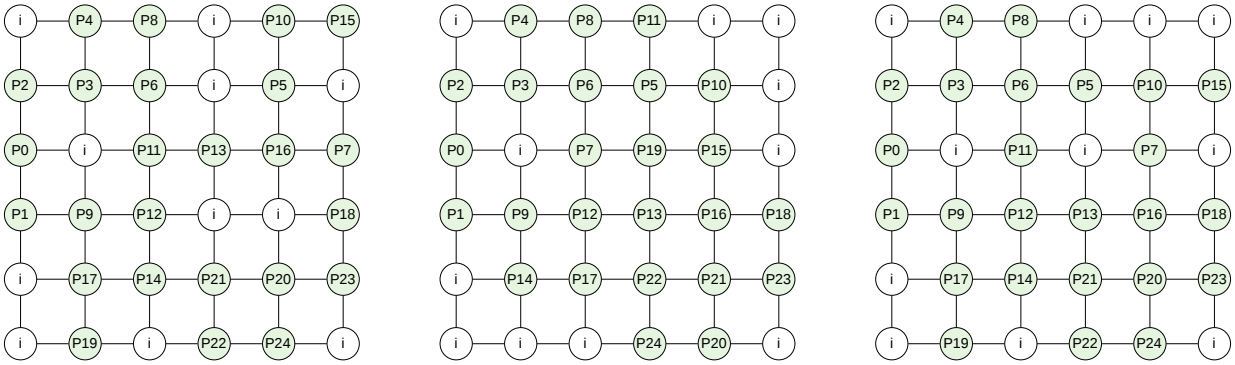
In order to assess the reliability of the evolved task maps and to determine how suitable the fault tolerance metric is for generating fault tolerant mappings, a set of experiments were carried out where single faults were injected into the task maps on the non-dominated Pareto front, and the impact on both performance and fault-tolerance observed.

1) Single Fault Post-Recovery Evaluation: Once a fault occurs, the graceful degradation mechanism is employed, and affected processing tasks are migrated from faulty to idle cores. The task map represented in Figure 4(a) – the same highly fault tolerant evolved individual illustrated in Figure 3(a) – has a *core fault* injected into the fifth node of the second row. Figure 4(b) shows how the affected task is migrated to the nearest idle core as part of the graceful degradation mechanism. In Figure 4(c) the original point of the Pareto front is highlighted with a blue circle along with the recalculated performance and fault tolerance metrics for the fault-recovered task map. The same fault is injected into each of the task maps on the non-dominated Pareto front, and their performance and fault tolerance metrics recalculated as shown in Figure 4(c). Following injection of the fault, the two most fault tolerant task maps exhibit an improvement in performance while becoming less fault tolerant to any further faults. In contrast, the least fault tolerant solutions suffer a negative impact on performance as well as suffering a fall in fault tolerance.

2) Single Fault Resilience: Since each task map on the non-dominated Pareto front represents a different mapping, it is expected that faults will affect each task map differently. In order to get a sense of the general fault resilience of a given task map, each possible fault was applied to the task map and the response, in terms of the effect on the fault tolerance and performance objectives, was calculated.

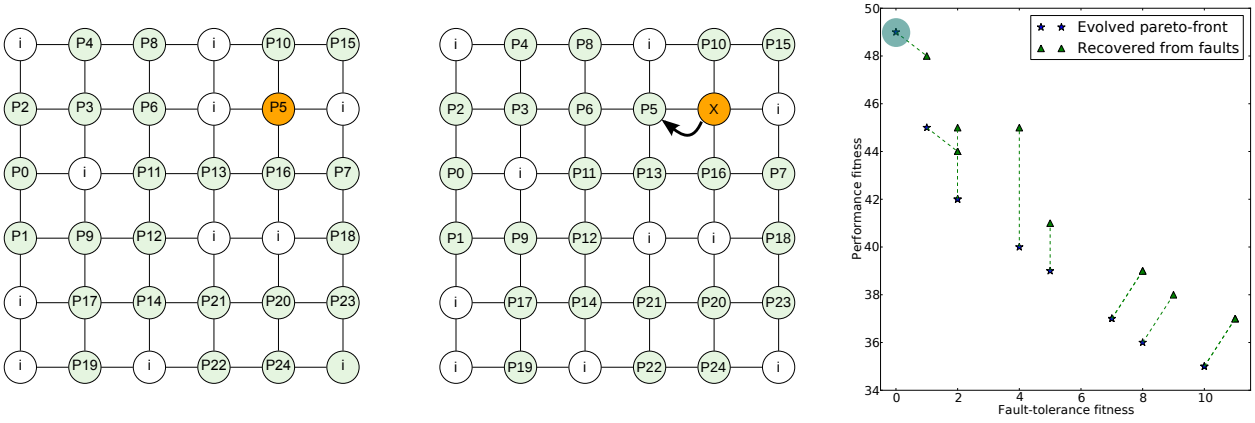
It is expected that the most fault tolerant task maps from the evolved Pareto front will exhibit a smaller average shift in performance in response to each possible fault. Additionally, the distribution of performance shifts from injected faults should be narrower for more fault tolerant task maps.

Figure 5 illustrates the distribution of performance shifts resulting from a *core fault* injection to each of the 36 nodes



(a) (b) (c)

Fig. 3. Three task maps chosen from the non-dominated Pareto front, exhibiting (a) good fault tolerance with poor performance, (b) good performance and poor fault tolerance (c) a balance between the two metrics.



(a) (b) (c)

Fig. 4. Graceful degradation mechanism being triggered on a task map after a core fault occurs, and corresponding impact on metrics. (a) task map with fault on node processing P5, (b) shows the task migration to an idle core, and (c) shows the shift in metrics for each of the individuals in the Pareto front once the same core fault is injected.

in the array. Each notched box plot represents the distribution of 36 performance points, each connected to a different core fault. The non-faulty performance is represented by a yellow star for each of the evolved Task Maps on the Pareto front. A similar plot is drawn in Figure 6 for *routing node* faults, injected into each of 36 routing nodes on the array.

The same multi-objective evolution was carried out for another 25 node graph, more sparsely connected than the one illustrated in Figure 1(a). Both the resulting Pareto front and the single-objective engineered task map were injected with core faults, and the resulting distribution of performance shifts can be examined in Figure 7. Although the multi-objective evolution resulted in a more sparsely populated Pareto front, the results are similar to the original 25-node graph, in that both median performance shift and fault-induced performance variance tend to increase as we evaluate task maps that are less fault tolerant. Additionally, the single-objective engineered task map still exhibits the worst variance in performance shift of the set.

In order to carry out a preliminary investigation on the impact of this approach in larger APGs, an evolutionary runs were carried out for a 26-node APG and a 27-node APG, along with the same fault injection procedure, and the results of these experiments are plotted in Figures 8 and 9, respectively.

Once again, the same trend is observed: the better performance a task map achieves, the more likely and of greater magnitude will be the impact on performance, in the presence of a fault. This observation validates the initial assumption that performance and fault-tolerance, as defined in this work, are opposing evolutionary pressures.

VI. Conclusions & Future Work

This work set out to use multi-objective evolution to generate task maps with a good trade-off between performance and fault tolerance. To evaluate the discovered solutions, a

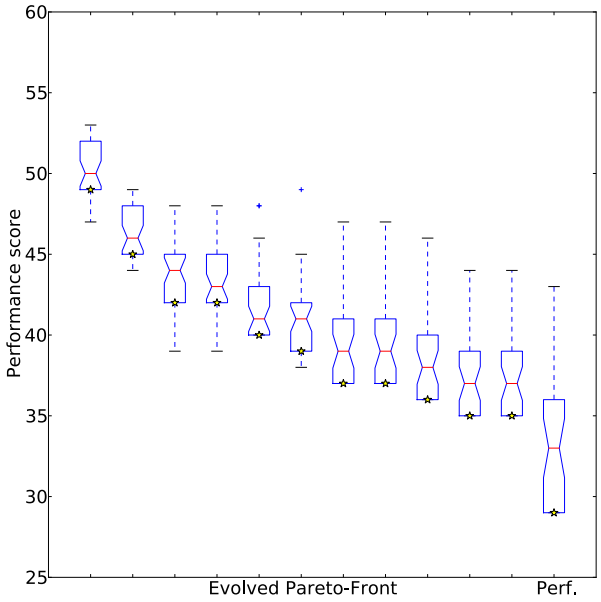


Fig. 5. Analysis of all possible core faults injected into each task map on the evolved non-dominated Pareto front (box plots 1-11 from the left) and to the task map resulting from a single objective evolutionary run focusing exclusively on performance (box plot on the far right), for the APG illustrated in Figure 1(a). The pre-fault performance metric is plotted as a star for each box plot.

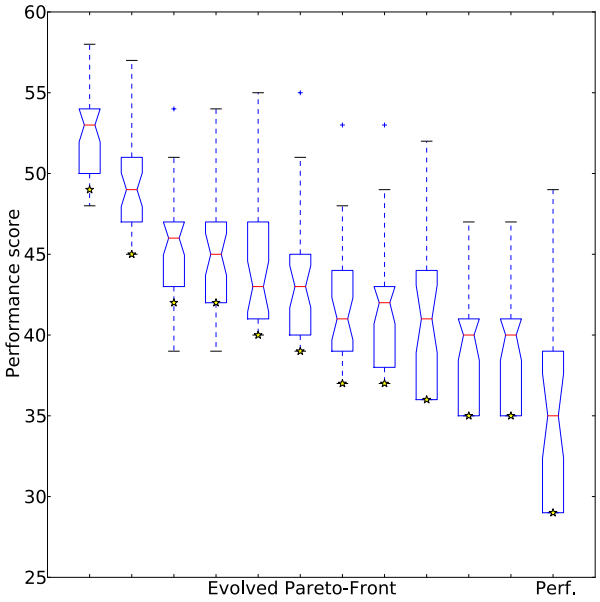


Fig. 6. Analysis of all possible routing node faults injected to each task map on the non-dominated Pareto front (box plots 1-11 from the left) and to the task map resulting from a single objective evolutionary run focusing exclusively on performance (box plot on the far right), for the APG illustrated in Figure 1(a). The pre-fault performance metric is plotted as a star for each task map.

full set of faults were injected to determine the distribution of the impact on performance caused by these faults.

Work is in progress to refine the performance metric to take into account the network traffic generated between pairs of communicating tasks. Additional performance criteria such as voltage, operating frequency and power consumption of individual cores will also be considered.

This work is also part of a larger investigation which deals with online optimization and re-mapping. The mechanisms described in this work are part of the *graceful degradation* approach, but there is also scope for *graceful amelioration*. The next stage of this research will, after a fault occurs, re-run the multi-objective optimization to find a new set of non-dominated Pareto front task maps. The new mappings, which are expected to be an improvement over the repaired mappings resulting from graceful degradation, can be used to improve the performance to give graceful amelioration. The process of fault, graceful degradation, re-evolution and amelioration can continue providing an adaptive on-line optimization while there are sufficient healthy cores to satisfy the requirements for a given APG.

This work is a part of research whose endpoint is a multi-region model, where each region contains a monitor that makes decisions about its own region while co-operating with adjacent regions to distribute work across the whole many-core array. It is assumed that optimal mappings within individual regions can combine to produce optimal behaviour across the multi-region many-core fabric. By this approach this work will be applied to a full many-core system consisting of 1000's of cores.

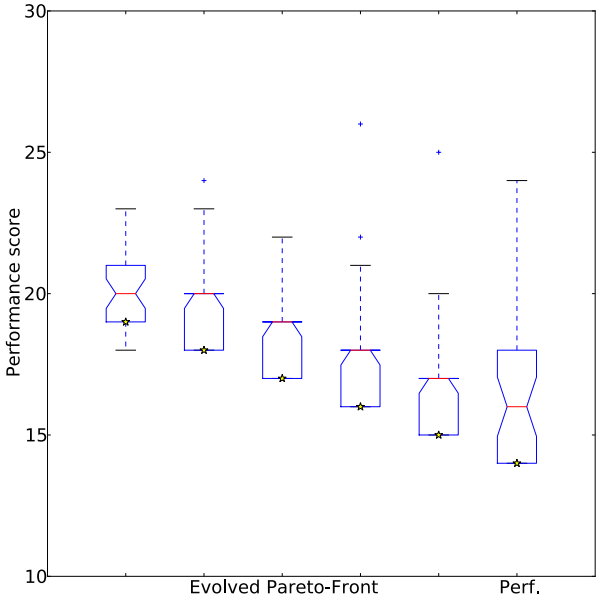


Fig. 7. Full sweep of core faults injected into each individual in the Pareto front and into the task map resulting from a single objective evolutionary run focusing exclusively on performance (last box plot on the right), for a sparsely connected 25-node APG. The pre-fault performance metric is plotted as a star for each task map.

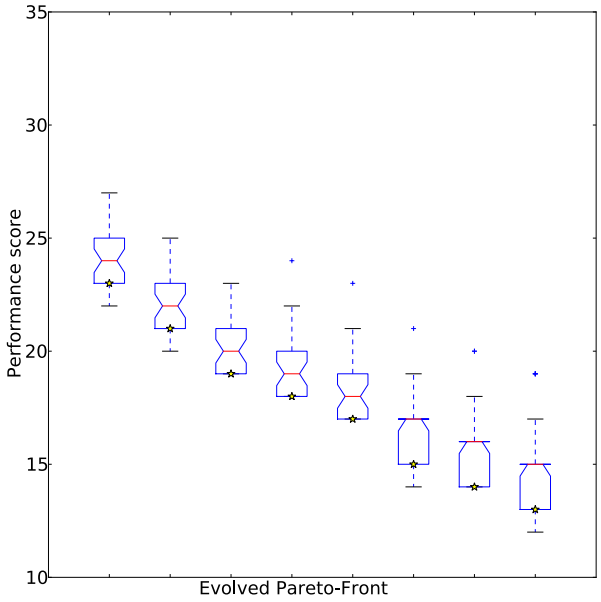


Fig. 8. Full sweep of core faults injected into each individual in the evolved Pareto front for the task map of a 26-node APG. The pre-fault performance metric is plotted as a star for each box plot.

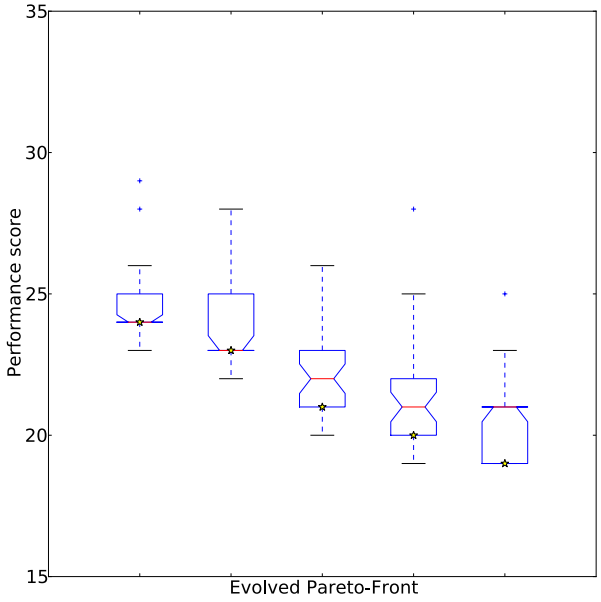


Fig. 9. Full sweep of core faults injected to both each individual in the Pareto front for the Task Map of a 27-node APG. The pre-fault performance metric is plotted as a star for each box plot.

ACKNOWLEDGEMENTS

The authors would like to thank EPSRC for their support through the Graceful project grant (EP/L000563/1) of which this work is part of.

REFERENCES

[1] S. Borkar, "Thousand Core ChipsA Technology Perspective," *Design Automation Conference. DAC '07. 44th ACM/IEEE*, pp. 746–749, 2007.

[2] M. B. Taylor, "A Landscape of the New Dark Silicon Design Regime," *Micro, IEEE*, vol. 33, no. 5, pp. 8–19, 2013.

[3] J. Hu and R. Marculescu, "Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints," *Design Automation Conference, ASP-DAC 2003. Asia and South Pacific*, pp. 233–239, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=1119818

[4] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," *Design, Automation and Test in Europe Conference and Exhibition, 2004.*, pp. 0–5, 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=969207

[5] T. Lei and S. Kumar, "A Two-step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," in *Euromicro Symposium on Digital System Design (DSD'03).*, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1231923

[6] G. Ascia, V. Catania, and M. Palesi, "Multi-objective Mapping for Mesh-based NoC Architectures," *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004.*, pp. 182–187, 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1016765

[7] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli, "Run-Time Mapping for Reliable Many-Cores Based on Energy/Performance Trade-offs," *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium*, pp. 58–64, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6653583

[8] Z. Lei, H. Yinhe, L. Huawei, and L. Xiaowei, "Fault Tolerance Mechanism in Chip Many-Core Processors," *Tsinghua Science & Technology*, vol. 12, no. July, pp. 169–174, 2007.

[9] C. Lee, H. Kim, H.-w. Park, S. Kim, H. Oh, and S. Ha, "A task remapping technique for reliable multi-core embedded systems," *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES/ISSS '10*, p. 307, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1878961.1879014

[10] C.-L. Chou and R. Marculescu, "FARM: Fault-aware resource management in NoC-based multiprocessor platforms," *2011 Design, Automation & Test in Europe*, pp. 1–6, mar 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5763113

[11] Q. Le, G. Yang, W. N. N. Hung, X. Song, and X. Zhang, "Pareto optimal mapping for tile-based network-on-chip under reliability constraints," *International Journal of Computer Mathematics*, vol. 92, no. 1, pp. 41–58, may 2014. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00207160.2014.892073

[12] D. S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[13] R. Michael and S. David, *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

[14] K. Deb, S. Pratab, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NGSA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[15] P. Campos, N. Dahir, C. A. Bonney, M. A. Trefzer, A. M. Tyrrell, and G. Tempesti, "XL-STaGe: A Cross-Layer Scalable Tool for Graph Generation, Evaluation and Implementation," in *IEEE Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2016*, 2016.

[16] M. Ebrahimi, M. Daneshtalab, J. Plosila, and F. Mehdipour, "MD: Minimal path-based fault-tolerant routing in on-Chip Networks," in *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Ieee, jan 2013, pp. 35–40. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6509555

[17] J. Wang, X. Wang, L. Huang, T. S. Mak, and G. Li, "A Fault-tolerant Routing Algorithm for NoC Using Farthest Reachable Routers," in *Proceedings - 2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing, DASC 2013*, 2013, pp. 153–158.