Artificial bee colony-inspired run-time task management for many-core systems

Ali Abuassal, Gianluca Tempesti, Martin A. Trefzer
Deptartment of Electronic Engineering, The University of York, York, UK
ali.abuassal, gianluca.tempesti, martin.trefzer@york.ac.uk

***Abstract*—Efficient resource and application management is one of the most complex and challenging tasks in high performance computing. Large-scale computing systems that contain hundreds, thousands or even millions of cores demand solutions that can operate in a distributed, robust, and scalable fashion. However, while hardware parallelism is relatively straight forward to achieve, this is not generally the case for software. This leads to under-utilization of the hardware parallelism as well as imbalanced load distribution causing inefficiency and hotspots. In response to this challenge, this paper introduces a novel distributed and decentralized run-time management algorithm. The proposed method is guided by an optimization model inspired by artificial bee colonies (ABC). While ABC have proven useful for optimizing large sets of numerical test functions, this is the first time they are applied in the context of many-core system management. The initial result shows that, the ABC model is promising in context of run-time management for many-core systems. It is also anticipated that the algorithms bio-inspired foundations will inherently enable scalability, reliability, and adaptation. We are showing initial experiments, where the initial results indicate the capability of our model to improve the thermal distribution across the system.**

***Index Terms*—Many-core systems, network-on-chip, bio-inspired, Artificial Bee Colony, run-time management.**

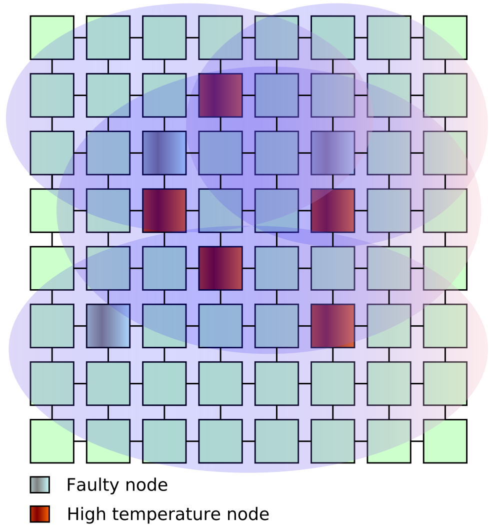


Fig. 1: Example of a *many-core* system based on a 2D mesh topology.

I. Introduction

The scaling of semiconductor technology has led to the integration of billions of transistors on a single chip. As Moore predicted, the number of transistors on a single chip has so far roughly doubled every two years since the 1970's [1]. This increase in chip density, however, is anticipated to be reaching its end due to physical limitations, particularly in terms of frequency and due to the difficulty of dissipating heat. *Many-core* systems (systems consisting of large numbers of processor units operating in parallel using only local clocks for synchronization, as shown in Figure 1) are generally seen as an opportunity to increase computational performance within these physical constraints. By moving to many-core, issues with clock speeds can be avoided by effectively operating in a GALS (globally asynchronous locally synchronous) regime, where heat can be better managed since processors in the system are able to run at lower frequencies (or sometimes be stopped if required).

Current trends strongly point at on-chip many-core system architectures. Examples include Intel's SCC [2], Tilera's TILE-Gx family [3] and SpiNNaker [4].These architectures feature large sets of cores which are connected through a network-on-chip (NoC). Chips with thousands, or even millions, of cores appear to be a technology perspective that is expected to become a reality in the near future. Hence, the problems of scalability and resource management are real and of significant relevance: if no new paradigms or algorithms are developed, complex future many-core systems will suffer from poor efficiency because they will tend to spend a large amount of their communication and computation capabilities to manage their own resources, which will lead to the problem of resource allocation.

In the future, millions of cores might be connected to form a single computing system (many-core system). This means that, in order to still be able to efficiently use such computing environments, scalable approaches need to be provided which can handle these very large-scale systems. Thus, scalability has become one of the most important factors for resource management for future computing systems. Decentralized methods are generally the approaches that lead to maximum scalability, since centralized models commonly suffer from several inherent disadvantages such as communication bottlenecks and single points of failure. On the other hand, implementing purely decentralized resource management models might significantly increase the level of complexity (and in some cases, make the approach infeasible) [5]. Distributed run-time management models that are guided by bio-inspired (in our case social-insect inspired) algorithms can be considered as realistic alternative decentralized models for resource management, since they can provide scalability while avoiding large increases in system complexity.

In order to map applications onto many-core systems, the applications need to be divided into individual tasks (known as task graph) that run on different cores. it is important to efficiently map the application tasks into the hardware resources. There are many task graph generation tools that have been proposed in the literature. This work utilizes XL-STaGe (which is a cross-layer tool for traffic-inclusive directed acyclic graph generation and implementation [6]) to generate task graphs.

The general problem of mapping and optimizing applications on cores can be expressed as the *quadratic assignment problem* [7]. The size of the search space for an optimal mapping grows factorially with the number of cores. Moreover, these mappings can not be efficiently predefined when the workload is highly dynamic. Consequently, they should ideally be handled dynamically at run-time.

Recently, resource management for many-core systems has been the focus of significant research [8] [9], and growth in the number of cores has encouraged the development of novel algorithms and methodologies to address this issue. Several application mapping algorithms that are aware of resource status have been recently proposed. [10] provides an intensive survey of the mapping methodologies targeting multi-core system for both homogeneous and heterogeneous architectures. A methodology for application-aware task mapping on Network-on-Chip based architectures is presented in [11]. The authors propose task allocation algorithms that consider the network congestion to reduce latency and to manage power consumption.

The remainder of this paper is organized as follows. Section 2 describes types of run-time management in many-core systems. Section 3 discusses the proposed hierarchical distributed management. Section 4 illustrates the bio-inspired model. While section 5 lays out the experimental set-up and results, section 6 concludes the work.

II. A Hierarchical Distributed Management Algorithm

The large number of cores in many-core systems increase the complexity of task mapping and system management. The main concerns in large systems include (i) scalability; (ii) dynamic workload; and (iii) reliability. Decentralising mapping and management decisions across the system is a necessity to ensure scalability. The workloads of emerging many-core systems are highly likely to be dynamic (i.e. new applications may start at any time), leading to different mapping scenarios. Accordingly, it is important to be able to fully or partially remap tasks at run-time to support a dynamic workload assignment. This also plays a crucial role in the reliability of many-core systems, e.g. imbalanced loads may introduce hotspot areas in the chip and cause thermal issues. As a result, distributed resource-aware applications could be a suitable management candidate to overcome the aforementioned issues. Distributed management (distributed decision making) often involves monitors, which are nodes that are dedicated to watching other nodes.

In this work, two types of monitors (physical and application) are introduced to shape a hierarchical distributed algorithm, providing a hybrid physical and logical monitoring mechanism. Figure 2 illustrates the mapping of two example application task graphs, consisting of 9 and 6 tasks respectively, onto a many-core platform based on the proposed algorithm and model.

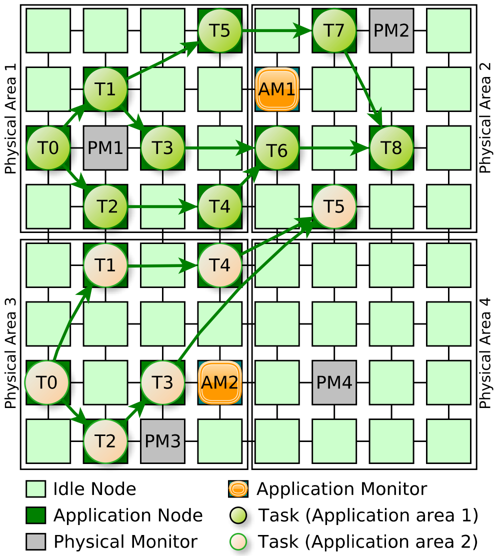


Fig. 2: Proposed distributed run-time management model when mapping two applications onto a many-core system. Nodes that are not executing any task are idle nodes, while application nodes are running tasks are active nodes.

In Figure 2, there are additional run-time management tasks mapped alongside the applications.

The *Physical Monitors (PMs)* are responsible for monitoring a fixed region of processing elements (PEs), referred to as *physical monitoring area*, the size of which is system dependent but assumed fixed for a given system. The PMs manage all the nodes within its zone (including active and idle nodes). The main functionalities of the PMs include managing task migration, power consumption, temperature, NoC router frequency, and enabling/disabling nodes. The PMs are allocated at design time.

An *Application Monitor (AM)* manages nodes that execute tasks belonging to the same application. The main functionality of AMs is to ensure the ensuring quality of services and manage task migration besides providing communication between PMs.

The size of the physical area is defined at design time, whereas the size of the application area varies at run-time. An application area might span multiple physical areas

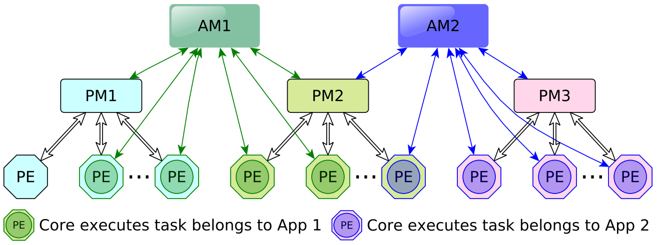

AM1
AM2
PM1
PM2
PM3
PE
Core executes task belongs to App 1
Core executes task belongs to App 2

Fig. 3: Communication between application nodes, physical monitors, and application monitor as well as the hierarchy of the algorithm. Two applications running on three different physical areas are shown.

and multiple applications can share the same physical area. Figure 3 illustrates the boundaries between the physical and application areas as well as the communication between the AMs, PMs and PEs in the case of two applications running on the system. In this model, the AMs communicate only with the active nodes (application nodes) that execute tasks belonging to their application. PMs, however, communicate with all nodes within their monitoring areas.

When a new application is about to start on the system, its AM is initiated on a random idle core. The randomly chosen initial core acts as a seed for finding resources for its application. The AM then searches for a suitable set of cores after communicating with the PMs.

For example in Figure 2, physical area 1 (PM1) runs tasks belonging to application 1, while PM3 executes part of application 2 only. The second physical area (managed by PM2) is shared between the two applications, as some nodes are executing tasks from application 1 and one node executes a task belonging to application 2. Once a core has been given a task and it is activated, a communication link will be established between this node and its AM.

To determine the mapping/remapping of an application tasks on a set of cores, we propose a bio-inspired model that is detailed in the next section.

III. Bio-Inspired Model

It is increasingly being recognised that bio-inspired algorithms are useful for addressing highly complex problems to achieve working solutions in short time, especially with highly dynamical problems (such as managing many-core systems).

Matthew Rowlings et.al. [12] propose an adaptive task allocation across many-core systems based on social insects and their decentralised nature to achieve high scalability in many-core systems. An optimization flow based on genetic algorithm to map applications into NoC-based platforms is shown in [13]. The results demonstrate the ability of the proposed approach in finding good mapping solutions in light of the optimization criterion.

In 2011, Karaboga introduced artificial bee colony (ABC) algorithm [14]. The artificial bee colony algorithm is based on modelling real bee behaviours in finding food sources (nectar) and sharing the information with the other bees in the hive. Honey bees share information by performing a special dance, known as the *waggle dance* [15] which provides information about the direction and distance to patches of flowers yielding nectar. Honey bees are social insects and live in large organized communities. Each kind of bee has specific skills and carries out specific tasks with the aim of facilitating the survival of the colony.

Karaboga models three bee behaviours in the colony: *employed bees*, associated with specific food sources; *onlooker bees*, watching the dance of employed bees within the hive; and the *scout bee*, which searches for food sources and shares the information with other bees.

Managing many-core systems can be inspired by the honey bee colony and how they maintain their colony with the available resources.

A comparative study into swarm intelligence algorithms for dynamic task scheduling in cloud computing has been carried out [16], which suggests that the ABC algorithm outperforms other algorithms in this context.

Karaboga and Ozturk [17] took the ABC model further and applied it to clustering tasks. They showed how ABC outperforms particle swarm optimization (PSO) for the same clustering tasks. They also experimented with the model for classification tasks and compared it with traditional classification algorithms such as Radial Basis Functions (RBF) and Bayesian Networks. The results proved the advantages of ABC model.

Scouting feature of honeybees is studied in [18]. Which concludes that swarms need to decide quickly on a new source, but not so quickly that it is likely to settle on a source of low quality when better sources are available or the current source has a lower cost. Moreover, the colony preserves some historical information related to food sources, causing them to be revisited as they become productive again.

These features can be applied to many-core systems: for example, if a core gets too hot either its frequency can be reduced or its task can be migrated to an idle core if the temperature of the core reaches some threshold. In addition to that, the model can memorize previous mapping, therefore, the best historical mapping can be reused or a hot core can be utilized when it cools down.

The ABC model is extended in [19] by adding inspector bees which can memorize the best solution across different cycles, when a source is abandoned by the colony and is not considered as the best solution for the next cycle, the inspector preserves this information. Furthermore, bees overcome the reduction in the amount of nectar on a food source (flower) by visiting it less often, eventually migrating to another food source if the amount of nectar falls below some threshold. In other words, bees compare the cost of moving (migrating) to other, distant food sources or keep visiting the same food sources but less often.

In this paper, we propose a dynamic multi-colony artificial bee (DMCAB) model, based on ABC. The DMCAB model consists of three essential components; employed bees, un-

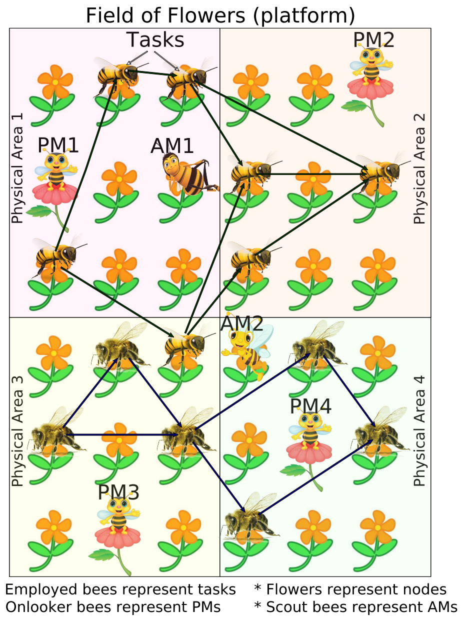

Field of Flowers (platform)
Tasks
PM2
Physical Area 1
PM1
AM1
Physical Area 2
AM2
Physical Area 3
PM4
Physical Area 4
PM3
Employed bees represent tasks
Onlooker bees represent PMs
* Flowers represent nodes
* Scout bees represent AMs

Fig. 4: The proposed dynamic multi-colony artificial bee (DMCAB) model. While tasks are presented by employed bees, physical and application monitors are presented by onlooker and scouts bees, respectively. Flowers reserve as cores.

employed bees, and food sources. Unemployed bees are further classified into onlooker and scout bees. Figure 4 shows a representation of the hardware system in the DMCAB model when two application are being executed. In which the application tasks are represented by employed bees. While physical and application monitors are represented by onlooker and scout bees respectively, food sources represent processing nodes. The first two components, employed and unemployed foraging bees, search for rich food sources, which is the third component, close to their hive. The model also defines two leading modes of behaviour which are necessary for self-organizing and collective intelligence; recruitment of employed bees to rich food sources and abandoning of poor sources.

Algorithm 1. DMCAB: algorithm's pseudo-code

```
Initialize employed, onlooker, and scout bees
Assigning employed bees to food sources randomly
While application(s) running: loop
 For all the applications
  For all the employed bees assigned to food sources
   Evaluate the nectar (fitness) of sources by onlooker
   If the nectar < the threshold then
      Search for new food source by scout
      Generate a new mapping
      Assigning employed bee(s) to new source(s)
      Memorize the abandoned sources
Go to loop
```

Onlooker bees (physical monitors) watch over fixed predefined areas. For instance, PM1 in Figure 4 watches physical area 1 only. Scout bees (application monitors), manage nodes that are running tasks belonging to its application. For example, AM1 oversees seven nodes that are executing application 1, which spreads across three physical areas. Application monitors, however, do not oversee idle nodes. In addition to their main task, application monitors provide communication between the physical monitors. This hierarchy provides a big picture of the application. While physical monitors manage the physical side of the system (such as temperature and power), the application monitors manage the logical part such as quality of service. In the DMCAB model, applications can dynamically start or terminate. They can also grow or shrink depending on the hardware resources and the required quality of service.

Pseudo-code for the DMCAB model is exhibited in algorithm 1, which illustrates the steps of the model. The goal of this model is to guide the distributed monitoring algorithm through the search space for optimal/near optimal mappings.

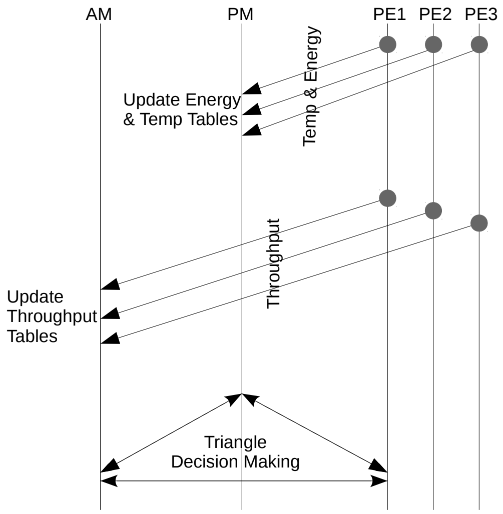


Fig. 5: Monitoring and decision making protocol.

Figure 5 represents the protocol of the monitoring and decision making. All the nodes update their status by sending their physical (temperature and power consumption) and logical (throughput) information to their physical and application monitors, respectively. A triangle communication between a specific node, physical and application monitors is established prior to voting when making a decision.

Three levels of decision making are considered, namely; node, physical monitors, and application monitors. Migrating a task will involve the three levels to vote, for instance, a physical monitor may detect a hot node and raise a flag to migrate the task from that node, but since the quality of service is met, the application monitor votes to keep the task running on the same node. In the third level, nodes usually vote be idle and migrate the tasks.

IV. Experiment and Results

The aim of the experiment presented in this work is to demonstrate how the temperature of a many-core system can be managed while keeping communication-to-computation cost (CCC) low. A thermal model was developed for this work to estimate changes in temperature of the nodes because actual measurement was not available.

A. Hardware Platform

The Graceful platform G0 (Figure 6) consists of 10 Xilinx ZC702 Zynq development boards, based on Zynq-7000 System-on-Chip devices. These boards consist of a processing system (PS) that includes a dual-core ARM Cortex-A9 processor, plus programmable logic, which is used to implement routing logic. Custom connection boards were designed to implement the interconnection network (NoC). The topology setup of the platform is shown in Figure 8.

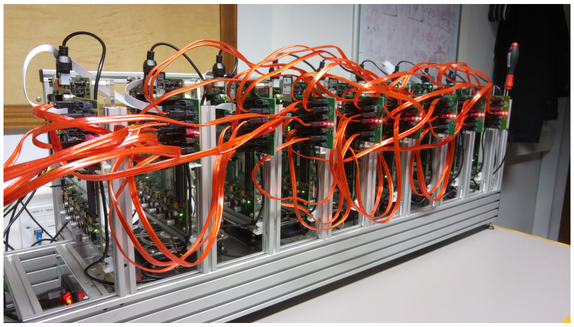


Fig. 6: A hardware platform of 10 Nodes based on Zynq-7000 family

B. Task Graph

Figure 7 displays a fork-join task graph typical of [20]. We considered this task graph to be load balanced (which is the process of spreading network traffic, computing workloads over a group of resources/nodes). Various load balancing strategies are discussed in [21] and [22].

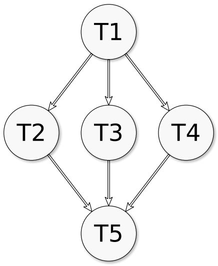


Fig. 7: Acyclic fork-join task graph from [20].

C. Thermal Model

A thermal model is executed on the physical monitor where the temperature of each node is estimated based on workload. It is assumed that the increase in core temperature (Equation 1) takes place during computation, according to the following formulas:

$$T = T_0 + \delta T^+ \tag{1}$$
$$\delta T^+ = K_h * (T_{max} - T_0) \tag{2}$$
$$K_h = R_c * f \tag{3}$$

Where:
δT^+ Increase in the temperature.
T_{max} Maximum temperature.
T_0 Previous temperature.
T Current temperature.
K_h Heating scaling factor.
f Frequency.
R_c Empirical constant in range of $(20 - 45) * 10^{-12}$. Its value is selected based on observation from a real chip to provide realistic temperature values.

The cooling, on the other hand, happens when there is no computation, i.e. a core is idle or waiting to receive data. Equation 4 and Equation 5 illustrate the cooling formula.

$$T = T_0 - \delta T^- \tag{4}$$
$$\delta T^- = K_c * (T_0 - T_{min}) \tag{5}$$

Where:
δT^- Decrease in the temperature.
T_{min} Minimum temperature.
T_0 Previous temperature.
K_c Cooling scaling factor.

The maximum temperature is considered to be 100°C which is the critical temperature of the chip. The minimum temperature, on the other hand, is set to 35°C as this is typical temperature of a chip in idle state. The cooling constants and frequency settings are the same for all nodes but heating constants depend on the empirical constant for each node (modelling variability) and the frequency, as in Equation 3. This means that the higher the frequency the larger the increase in the temperature. Generally speaking, chips heat up much faster than they cool down.

D. Communication-to-Computation Cost (CCC)

Communication-to-computation cost (Equation 6) plays a crucial role in the decision making, as application monitors rely on it to make a decision. In reality, bees tend to calculate the cost of distance against the amount and quality of food before migrating to another food source.

In order to calculate the CCC, some parameters need to be provided; the execution time for each task (computation time) and the cost of sending data from a node to another (communication cost). Table I illustrates the estimation of

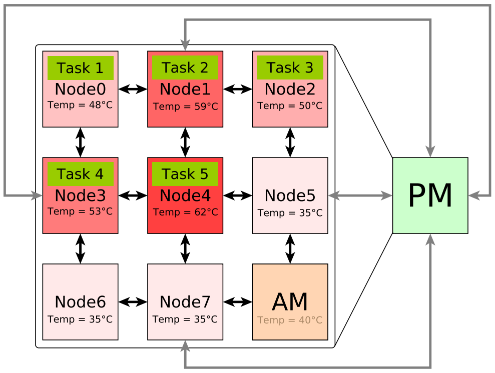


Fig. 8: An example temperature scenario of the 10-node system executing an application

the computation cost for each task. Communication cost, on the other hand, can be seen in Table II, for which we assume communication costs between any adjacent nodes is the same across the system. However, if the NoC router frequency is decreased, the communication cost will increase as demonstrated in Table II. The number of hops is also considered since they may introduce more latency.

$$\text{CCC} = \text{Comm}_{\text{cost}}/\text{Comp}_{\text{cost}} \tag{6}$$

TABLE I: Estimated computation cost for each task

Task	Time (ms)
Task1	27
Task2	12
Task3	12
Task4	12
Task5	35
Total	98

TABLE II: Communication Cost for different mappings based on various frequencies

Communication Cost @200MHz => 2, @100MHz => 4, @50MHz => 8				
	Mapping(a)	Mapping(b)	Mapping(c)	Mapping(d)
T1 to T2	C1	C1	C1	C1
T1 to T3	2*C1	2*C1	2*C1	2*C1
T1 to T4	C1	C1	C1	C1
T2 to T5	C2	2*C2	3*C2	2*C2
T3 to T5	2*C2	C2	4*C2	3*C2
T4 to T5	C2	2*C2	C2	2*C2
Total	4C1+4C2	4C1+5C2	4C1+8*C2	4*C1+7*C2
200MHz	= 16	= 18	= 24	= 22
100MHz	= 32	= 36	= 48	= 44
50MHz	= 64	= 72	= 96	= 88

TABLE III: Communication-to-computation cost based on given communication and computation costs at different NoC router frequencies

	NoC Router Frequency		
	200MHz	100MHz	50MHz
$\text{CCC}_{\text{mapping(a)}}$	0.16	0.33	0.65
$\text{CCC}_{\text{mapping(b)}}$	0.18	0.37	0.74
$\text{CCC}_{\text{mapping(c)}}$	0.25	0.49	0.98
$\text{CCC}_{\text{mapping(d)}}$	0.22	0.45	0.90

E. Results

For this experiment, we initially mapped an application of 5 tasks on the hardware platform (Figure 6), as shown in Figure 9. First we run the initial system without our run-time management, then we apply the proposed algorithm to minimize the temperature across the system while keeping low CCC.

The initial mapping is shown in Figure 9, in which the temperature of node 4 increases rapidly, creating a hotspot. On the other hand,Figure 10 illustrates the possible remapping when migrating task 5 if the algorithm decides to do so. The decision is made according to the voting in Table IV.

Figure 11 depicts temperature of the nodes without applying run-time management. Contrarily, Figure 12 illustrates the same system when we applied our proposed run-time management. The result shows that, in a system which has no dynamic run-time management the average temperature of the system was 69.64 °C, and peaked at 92.9 °C. By applying our runt-time management system the average temperature has dropped to 64.05 °C, and the peak temperature has dropped to 79.06 °C. This is an improvement as the temperature dropped by 8.03% and 14.9%, respectively,

V. Conclusion and Future Work

As many-core systems become large and extremely complicated, their reliability and scalability decreases. Moreover, their workload dynamically changes, and such dynamic workload scenarios require equally dynamic remapping/run-time mapping of the application. This work has introduced a bio-inspired run-time management algorithm based on hierarchical distributed physical and logical monitoring algorithms to improve resource allocation and robustness in many-core systems.

Initial results show that the proposed model is capable of improving the thermal distribution across the system, while keeping low communication-to-computation cost.

Our next step will be validating the proposed algorithm in hardware and testing the scalability by running large multiple applications. We are aiming to use the Graceful platform G1, which is a custom array of 64 nodes, connected via a custom network-on-chip (NoC). This many-core platform provides extensive monitoring and actuation facilities, providing a useful experimental platform for the proposed algorithms and methodologies.

Acknowledgements

This work was supported by funding from the Department of Electronic Engineering at the University of York and the Graceful project (EP/K040820/1) funded by EPSRC.

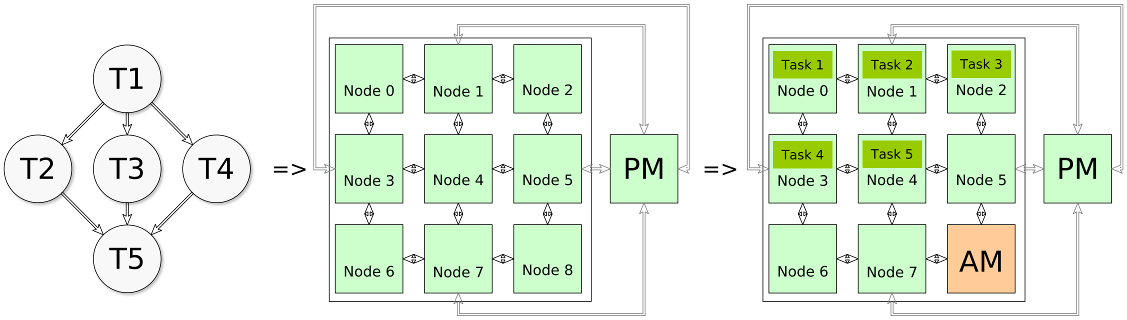
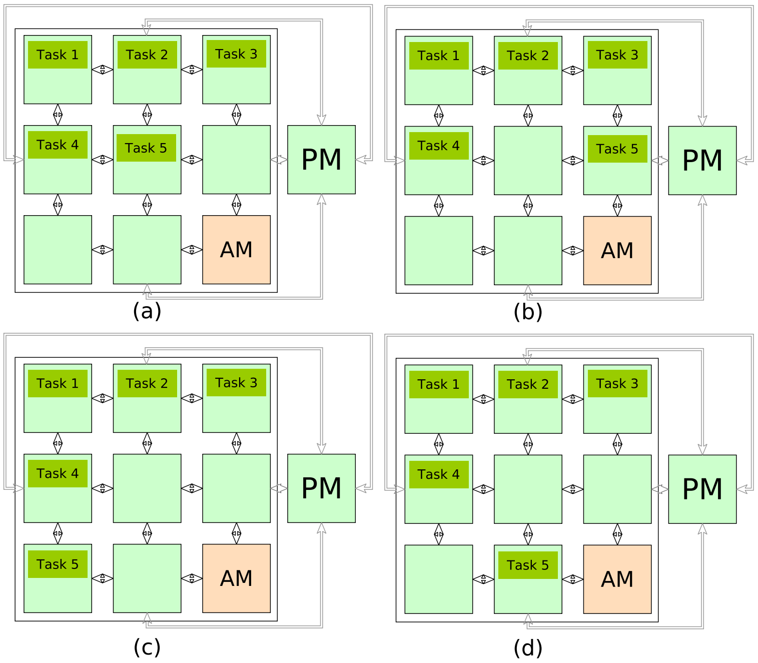


Fig. 9: One possible mapping of an application of 5 tasks onto a many-core system of 10 nodes



(a)
(b)
(c)
(d)

Fig. 10: (a) An example of an initial mapping and (b) to (d) possible remapping when task 5 is migrated

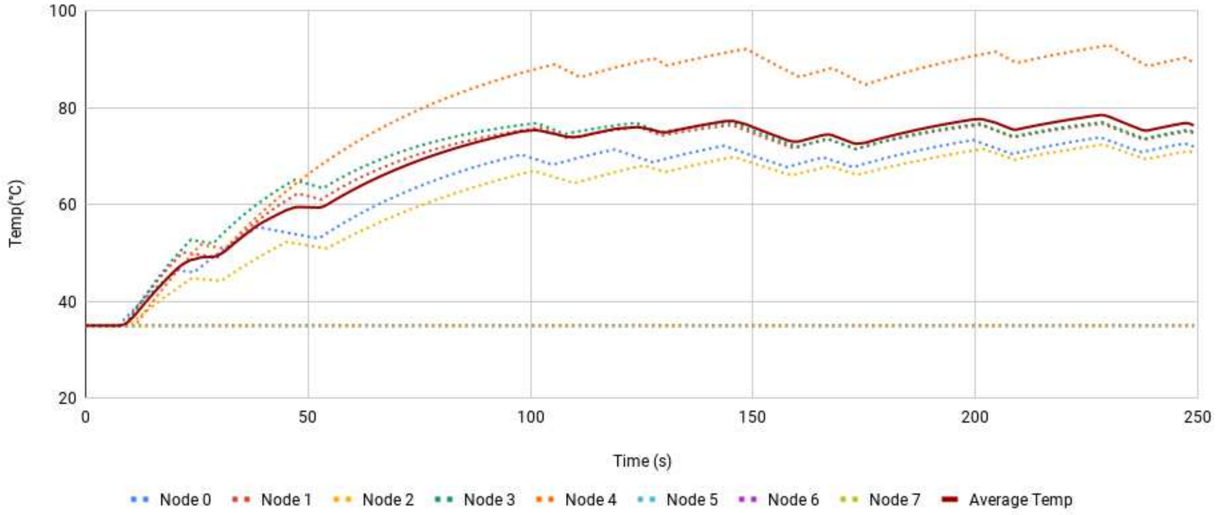


Fig. 11: Temperature of nodes when the system has no dynamic run time management

TABLE IV: decision making based on voting

	Decision on each level based on temperature and CCC								
Temperature (°C)	< 75			75 <=< 79			> 79		
CCC	< 0.40	0.40 <=< 0.60	> 0.60	< 0.40	0.40 <=< 0.60	> 0.60	< 0.40	0.40 <=< 0.60	> 0.60
Application Monitor	Stay	Scale⇑	Migrate	Stay	Scale⇑	Migrate	Stay	Scale⇑	Migrate
Physical Monitor	Stay	Stay	Stay	Scale⇓	Scale⇓	Scale⇓	Migrate	Migrate	Migrate
Node	Migrate	Migrate	Migrate	Migrate	Migrate	Migrate	Migrate	Migrate	Migrate
Decision	Stay	Scale⇑	Scale⇑	Scale⇓	Stay	Scale⇑	Scale⇓	Scale⇓	Migrate

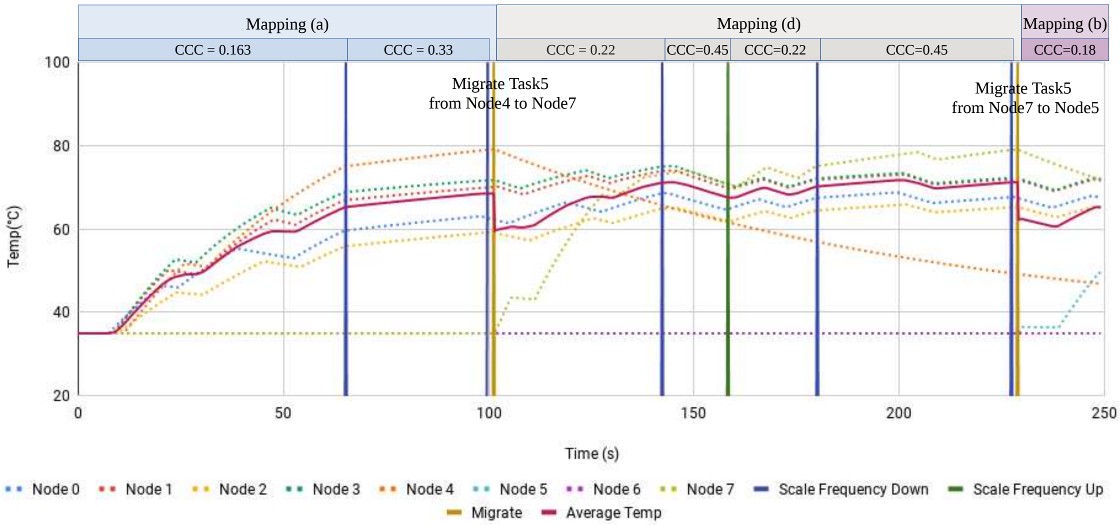


Fig. 12: Temperature of nodes managed by the proposed run-time management for mappings shown related to examples illustrated in Figure 10.

References

[1] G. Moore, "Cramming more components onto integrated circuits," *IEEE Solid-State Circuits Society Newsletter*, pp. 33–35, Sept 2006.

[2] J. Howard and S. Dighe, "A 48-Core IA-32, message-passing processor with DVFS in 45nm CMOS," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, February 2010, pp. 108–109.

[3] Tilera Corporation, "Tile-gx processor family," 2011.

[4] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, no. 5, pp. 652–665, May 2014.

[5] S. Kobbe and et.al, "Distrm: Distributed resource management for on-chip many-core systems," in *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct 2011.

[6] P. Campos, N. Dahir, C. Bonney, M. Trefzer, A. Tyrrell, and G. Tempesti, "XL-STaGe: A cross-layer scalable tool for graph generation, evaluation and implementation," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, July 2016, pp. 354–359.

[7] C. Marcon, E. Moreno, N. Calazans, and F. Moraes, "Evaluation of algorithms for low energy mapping onto NoCs." in *IEEE International Symposium on Circuits and Systems (IS-CAS)*, May 2007.

[8] B. M. Al-Hashimi, "Hardware reliability of embedded systems: Are we there yet?" in *2013 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2013.

[9] K. R. Basireddy, E. W. Wachter, B. M. Al-Hashimi, and G. V. Merrett, "Workload-aware runtime energy management for HPC systems," in *International Workshop on Optimization of Energy Efficient HPC & amp; Distributed Systems (OPTIM 2018)*, May 2018.

[10] A. K. Singh and et.al, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–10.

[11] C. Chou and R. Marculescu, "User-aware dynamic task allocation in networks-on-chip." *Automation and Test in Europe*, August 2008.

[12] M. Rowlings, A. M. Tyrrell, and M. A. Trefzer, "Social-insect-inspired adaptive task allocation for many-core systems," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, July 2016.

[13] J. V. Bruch and et.al, "Deadline, energy and buffer-aware task mapping optimization in noc-based socs using genetic algorithms," in *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*, Nov 2017, pp. 86–93.

[14] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," in *Technical Report-TR06*, October 2005.

[15] C. Grüter, M. S. Balbuena, and W. M. Farina, "Informational conflicts created by the waggle dance," vol. 275, pp. 1321–1327, 2008.

[16] G. Elhady and M. Tawfeek, "A comparative study into swarm intelligence algorithms for dynamic tasks scheduling in cloud computing," in *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, Dec 2015, pp. 362–369.

[17] D. Karaboga and C. Ozturk, "A novel clustering approach: Artificial bee colony (abc) algorithm," in *Applied Soft Computing*, 2011.

[18] S. Janson, M. Middendorf, and M. Beekman, "Searching for a new homescouting behavior of honeybee swarms," *Behavioral Ecology*, vol. 18, no. 2, pp. 384–392, 2007.

[19] C. "Birtolo, G. Capasso, D. Ronca, and G. Sorrentino, "Modeling an artificial bee colony with inspector for clustering tasks," in *Evolutionary Computation in Combinatorial Optimisation*, C. Blum and G. Ochoa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 182–193.

[20] B. Sethuraman and R. Vemuri, "optimap: a tool for automated generation of noc architectures using multi-port routers for fpgas," in *Proceedings of the Design Automation Test in Europe Conference*, vol. 1, March 2006.

[21] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9, Sept 1993.

[22] J. Doe, "Load balancing strategies in parallel computing : Short survey," 2015.