International Journal of Electronics
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713599654

Heterogeneous network-on-chip design through evolutionary computing

Ozcan Ozturk[a]; Dilek Demirbas[a]
[a] Computer Engineering Department, Bilkent University, Bilkent, Ankara, Turkey

Online publication date: 06 October 2010

To cite this Article Ozturk, Ozcan and Demirbas, Dilek(2010) 'Heterogeneous network-on-chip design through evolutionary computing', International Journal of Electronics, 97: 10, 1139 — 1161
To link to this Article: DOI: 10.1080/00207217.2010.512020
URL: http://dx.doi.org/10.1080/00207217.2010.512020

International Journal of Electronics
Vol. 97, No. 10, October 2010, 1139–1161

Taylor & Francis
Taylor & Francis Group

Heterogeneous network-on-chip design through evolutionary computing

Ozcan Ozturk* and Dilek Demirbas

Computer Engineering Department, Bilkent University, 06800 Bilkent, Ankara, Turkey

(*Received 4 November 2009; final version received 13 April 2010*)

This article explores the use of biologically inspired evolutionary computational techniques for designing and optimising heterogeneous network-on-chip (NoC) architectures, where the nodes of the NoC-based chip multiprocessor exhibit different properties such as performance, energy, temperature, area and communication bandwidth. Focusing primarily on array-dominated applications and heterogeneous execution environments, the proposed approach tries to optimise the distribution of the nodes for a given NoC area under the constraints present in the environment. This article is the first one, to our knowledge, that explores the possibility of employing evolutionary computational techniques for optimally placing the heterogeneous nodes in an NoC. We also compare our approach with an optimal integer linear programming (ILP) approach using a commercial ILP tool. The results collected so far are very encouraging and indicate that the proposed approach generates close results to the ILP-based approach with minimal execution latencies.

Keywords: NoC; evolutionary computing; genetic algorithm; heterogeneous

1. Introduction

As commonly accepted, the current performance trajectory of doubling chip performance every 24–36 months can be achieved by the integration of multiple processors on a chip rather than through increases in the clock rate of single processors due to the power limitations present in processor design. Multicore architectures have already made their way into the industry (Kahle et al. 2005; Kongetira et al. 2005; McGowen 2005; AMD Athlon 64 X2 Dual-Core Processor for Desktop, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9485_13041,00.html; Intel quad-core Xeon. http://www.intel.com/quad-core/?cid=cim:ggl—xeon_us_clovertown—k7449—s), with more aggressive configurations such as the Intel's 80 core TeraFlop being prototyped (http://www.intel.com/idf/). Since future technologies offer the promise of being able to integrate billions of transistors on a chip, the prospects of having hundreds of processors on a single chip along with an underlying memory hierarchy and an interconnection system are entirely feasible. Point-to-point buses will no longer be feasible after a certain number of nodes as the communication requirements between nodes will exponentially increase with the number of processors. A viable interconnection system shown to be

*Corresponding author. Email: ozturk@cs.bilkent.edu.tr

ISSN 0020-7217 print/ISSN 1362-3060 online

DOI: 10.1080/00207217.2010.512020
http://www.informaworld.com

promising for these future chip multiprocessors (CMPs) is network-on-chip (NoC) since it provides scalable, flexible and programmable communication. With this NoC-based CMP as the computing platform, a very rich set of research challenges arises. Circuit and architectural challenges such as router design, IP placement and sensor placement are currently being studied in both industry and academia. In comparison, the work on heterogeneous alternatives for these architectures has received considerably less attention. Motivated by this observation, we propose an evolutionary computational technique for the design of heterogeneous NoC architectures. More specifically, we implement a genetic algorithm (GA) to optimally select processors and their placements within the NoC architecture.

Evolutionary computing has been used as a powerful tool for solving many problems from different domains. Most of these problems are computationally intense multi-parameter optimisations, which can be solved in a methodology similar to how biological evolution functions. These tools use a repeated process over many generations through selection, mutation and cross-over, thereby exploring a larger design space. Complex systems can be modelled in a similar way in which living systems adapt to natural environments. Evolutionary computational designs have been shown to be very successful (Handl and Knowles 2007; Wang and Dang 2007), which makes them a very attractive tool for our heterogeneous NoC design problem. Our problem fits well into the evolutionary computing domain since the design space is very big for a brute-force computational method or an optimisation framework such as integer linear programming (ILP).

Even though complex processors provide higher single-thread performance, they consume more area and power compared to their simpler counterparts. Every application has a different processing need and a memory requirement. Even one single application has different requirements throughout its execution. An application may exploit a high level of instruction-level parallelism where a powerful processor will be a better match, whereas a simpler processor will suffice for a different application with lower instruction-level parallelism. To choose the best match for an application will reduce the power consumption. However, homogeneous CMPs provide only one type of processor to match all these various requirements. When Alpha processors EV4 and EV8 are compared, one can see that EV8 consumes 80 times more area and 12 times more power to gain just above two times more performance (Kumar, Tullsen, Ranganathan, Jouppi and Farkas 2004; Kumar, Tullsen, Jouppi and Ranganathan 2005). The marginal benefit provided by larger processors is lower compared to smaller processors, consequently one should try to utilise the available chip area and power budget according to the application needs.

Figure 1 gives the power, performance and area comparison for Alpha cores. This example is only given for illustrative purposes, i.e. they are not the actual cores used in our experimental evaluation. Figure 1a shows the average power consumption of Alpha processors EV4, EV5, EV6 and EV8. Figure 1b gives the normalise instructions per cycle (IPC) values with respect to EV4, whereas Figure 1c indicates the normalised area consumption of the aforementioned processors (scaled to the same 10 nanometre feature size) (Kumar et al. 2004; Kumar et al. 2005). When EV4 and EV8 are compared one can see that EV8 consumes 80 times more area and 12 times more power to gain just above 2 times more performance. The key observation from these charts is that the marginal benefit provided by larger cores is lower compared to smaller cores, consequently one should try to utilise the available chip area and power budget according to the application needs.

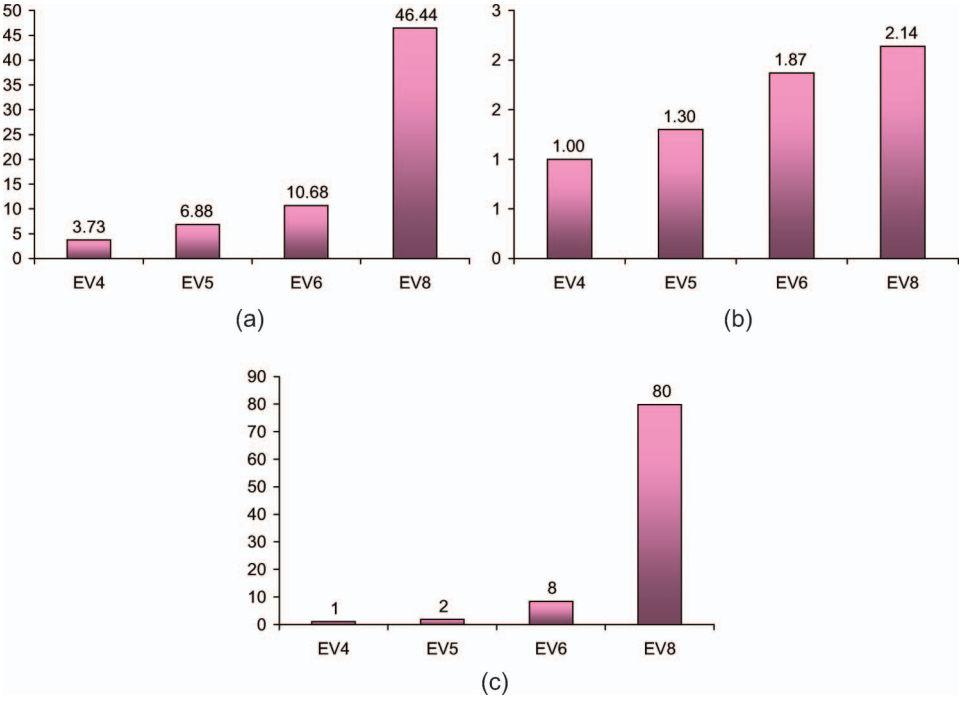

Figure 1. Comparison of Alpha cores EV4, EV5, EV6 and EV8. (a) Average power consumption in watts. (b) Performance comparison (IPC - instructions per cycle) with respect to EV4. (c) Relative sizes of cores normalised with respect to EV4.

The ability to dynamically switch between different processors and power down unused processors is the key in heterogeneous chip multiprocessing. It allows the processor to better match execution resources of application needs which in turn enables different workloads from high to low. It was shown that a representative heterogeneous processor using two processor types achieves as much as a 63% performance improvement over an equivalent-area homogeneous processor (Kumar et al. 2004; Kumar et al. 2005).

To overcome the limitations due to the homogeneous behaviour of CMPs, heterogeneous (asymmetric) CMPs have been proposed (Flachs et al. 2005; Pham et al. 2005). In comparison, NoC-based heterogeneous CMPs have received considerably less attention. The focus of this article is to perform the node placement for a given NoC-based heterogeneous CMP with various types of nodes under certain constraints. Our goal in selecting the location of each node is to minimise the communication distance and load, i.e. placing the frequently communicating nodes as close as possible. In addition to this, we try to select the most suitable processor to run on a node.

This approach has been implemented by both using a GA and an optimal ILP-based approach. This article is the first one, to our knowledge, that explores the possibility of employing evolutionary computational techniques for optimally placing the heterogeneous nodes in an NoC. ILP-based formulation has been implemented to show the accuracy of our GA-based technique.

The remainder of this article is structured as follows. Section 2 gives the related work on evolutionary computational methods and NoC-based heterogeneous CMPs. Section 3 gives details of our approach and introduces the NoC-based CMP abstraction. We present our evolutionary framework in section 4. The details of our ILP-based approach are given in section 5, and an experimental evaluation is presented in section 6. The article is concluded in Section 7.

2. Related work

We present the related work in three parts. First, we summarise the related work on NoC-based CMPs in general and the related studies on heterogeneous CMPs. Second, we discuss the related work on NoC-based heterogeneous CMPs. Finally, we explore the related studies on evolutionary-based NoC studies.

Prior research studied the concept of NoC-based CMPs (Benini and De Micheli 2001; Dally and Towles 2001), which aims at replacing the traditional interconnects with regular networks. These efforts are mostly motivated by increasing on-chip wire delays. Meanwhile, both academia and industry are witnessing the increasing popularity of CMPs (Taylor et al. 2002; Kahle et al. 2005; Kongetira et al. 2005; http://www.intel.com/idf/). With the trend towards increasing the number of on-chip processors, NoC-based CMPs will be an excellent interconnection choice for large-scale multi-core architectures due to their scalability, flexibility, performance and other advantages.

Several research groups have been working on modelling and simulating NoC-based CMPs. In Madsen et al. (2003), a NoC-based CMP model based on the concepts of allocator, scheduler and synchroniser is proposed and evaluated. Gerstlauer (2003) discusses how abstract communications performed over a NoC-based CMP. There are several emerging NoC-based platforms, for example, OPNET (Bolotin et al. 2004), a commercial NoC simulator, and MPARM (Loghi et al. 2004), a cycle accurate platform, etc. It turns out that NoCs consume a significant portion of the overall chip power. This fact motivates the need to analyse and model the power consumed by NoCs. Orion (Wang et al. 2002) is a cycle accurate power-performance simulator. Similarly, another tool presented by Eisley and Peh (2004), is used for estimating the power-performance metrics of NoCs at a more abstract level than Orion.

Prior research on heterogeneous CMPs addressed various problems ranging from hardware aspects such as selecting the right type of processors to software solutions at the OS level. Kumar et al. (2004, 2005) discuss the benefits of heterogeneous CMPs from many angles. Blume, Feldkaemper and Noll (2005) present a model-based exploration method to support the design flow of heterogeneous CMPs. They implement cost models for design space exploration using several cost parameters such as performance and throughput. Balakrishnan, Rajwar, Upton and Lai (2005) explore the effects of heterogeneity on commercial applications using a hardware prototype. Both software (Liu and Chaudhary 2003; Shee, Erdos and Parameswaran 2006; Brisolara et al. 2007) and hardware (Lieverse, Wolf, Vissers and Deprettere 2001; Ghiasi, Keller and Rawson 2005; Kumar, Tullsen and Jouppi 2006; Yan and Zhang 2007) solutions have been proposed to exploit heterogeneity in CMPs. There are only a handful of prior studies which focus on heterogeneity-aware NoC-based CMP design. Most of these studies consider heterogeneity only within the network domain, that is, they discuss mixing different types of router architectures and use an

optimised mix of these routers (Cardoso et al. 2005; Kreutz et al. 2005; Ahonen and Nurmi 2007).

On the other hand, evolutionary computational designs have been very successful (Handl and Knowles 2007; Wang and Dang 2007) in different domains, including VLSI design and layout (Schnecke and Vornberger 1996; Mazumder and Rudnick 1999). Ascia, Catania, Palesi and Parlato (2003) present a GA-based encoder to minimise switching activity on the communication bus. Palesi and Givargis (2002) use a GA to find pareto-optimal configurations from the design options in a parameterised SoC architecture. Apart from efforts on VLSI design, there are also approaches that specifically target NoC architectures. For example, Jena and Sherma (2007) address the problem of topological mapping of IPs on a NoC through a multi-objective GA. Lei and Kumar (2003) propose a two-step GA to map a task graph of a given application on to a NoC architecture. An application-specific NoC topology framework is presented by Leary, Srinivasan, Mehta and Chatha (2009). More specifically, a system-level floorplan is generated using a multi-objective GA. Srinivasan and Chatha (2005) present a GA-based technique for custom NoC topology design and application mapping. Leary et al. (2009) presented chaos genetic mapping, a mapping strategy for task graphs on NoCs. They try to improve the quality of service (QoS) in NoC architectures with a GA-based implementation.

Lambrechts et al.'s (2005) is one of the studies that primarily focuses on processor heterogeneity within NoC-based CMP design. In this work, the authors present a power assessment of a realistic heterogeneous NoC-based CMP platform which primarily targets the video processing chain from camera to display. The main difference between this work and ours is that we propose a GA to reduce the overall communication and computation through efficient placement and processor selection. Ascia, Catania and Palesi (2004) proposed a GA-based multi-objective NoC mapping scheme, where the NoC is composed of $n \times m$ tiles of equal size. They try to optimally select the placement of fixed IP blocks within the given tiles based on both power and performance. Similarly, da Silva, Nedjah and deMacedoMourelle (2009) propose a methodology to topologically map the selected IPs given as an application characterisation graph (APG). More specifically, they describe applications using APG, where nodes indicate individual IPs and edges represent the communication between IPs. Zhou, Zhang and Mao (2006) use a similar GA-based technique to perform a placement on the NoC architecture with different objectives. They try to minimise the communication overhead by reducing the hop count while keeping the thermal balance. As compared to these studies, in our approach, we do not assume a fixed set of IPs or processors; rather we try to select the optimum set of processor types for a given application. In addition to this selection, we also perform a placement within the available on-chip area. Moreover, these techniques are mostly concerned with the communication effects since the main task is to map the IPs or processors within the NoC. However, in our combined optimisation problem (selection + mapping), we also consider computation and area in addition to communication.

3. Overview of our approach

Figure 2 illustrates the high-level view of our approach. The input code (after parallelisation and mapping) is fed to a compiler analysis module which identifies the

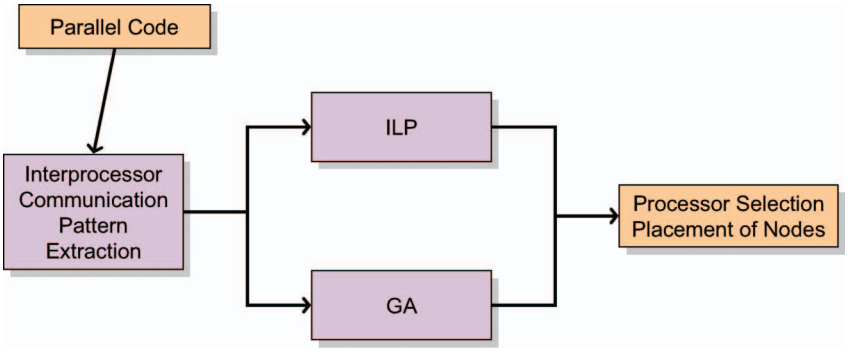

Figure 2. High-level view of our approach.

set of CMP nodes that communicate with each other. This information is subsequently passed to the solver which determines the location of each node within the NoC-based CMP and the type of processor used for each node. Specific solvers used are a GA-based solver implemented using Java and an ILP-based solver implemented on a commercial tool. Our goal in selecting the location of each node is to minimise the communication load. Note that, depending on the functionality, each node can exhibit different characteristics in terms of performance, energy, temperature, area and communication bandwidth supported by the switch available within the node. This information is crucial for selecting the type of processor.

3.1. *Target NoC-based CMP architecture*

In our proposed approach, the NoC-based heterogeneous CMP architecture is exposed to the compiler. This means the compiler accesses the state of the processors as well as the network switches, and manages the data/code movement between different components. Figure 3 shows the high-level view of a heterogeneous NoC-based CMP with a two-dimensional (2D) mesh topology. Each node of this mesh consists of a network switch/router (represented by *R*), a processor (represented by *CPU*) and a memory hierarchy (represented by *MH*). Except for boundary nodes, the network switch is responsible for direct communication with the neighbouring switches (i.e. north, south, west and east). Location of a node within this NoC-based CMP architecture can be represented using x and y coordinates. In our implementation, we use these coordinates to calculate the communication distances between the nodes. For example, if node i is at (x_i, y_i) and node j is at (x_j, y_j), we express the communication distance d for these two nodes with the Manhattan distance, that is, $d=|x_i-x_j| + |y_i-y_j|$. For each pair of adjacent nodes, i and j, the communication links between them are bi-directional. That is, there exists a communication link from node i to node j as well as a link from node j to node i.

3.2. *Interprocessor communication extraction*

As shown in Figure 2, we first need to extract the interprocessor communication information for a given application. For a given set of NoC-based CMP nodes, our

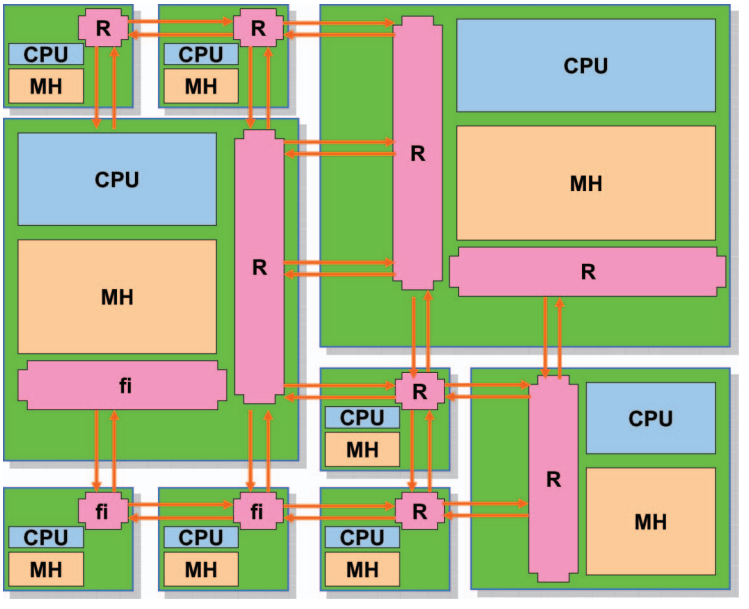

Figure 3. NoC-based heterogeneous CMP architecture.

eventual goal is to distribute the computation and data among the nodes using an optimising compiler.

In order to distribute the data among the processors, we need two types of information: the private data accessed by each processor and the amount of data shared across the processors. We obtain these through representing the data accessed by each processor and each processor pair using Presburger formulas and counting them. When Presburger formulations are generated, we perform the code to count the number of elements in each set. The resulting numbers (for private and shared data) are then fed to our data partitioning algorithm.

Similarly, we represent parallelisation information for a given nest using a vector referred to as the *parallelism vector* ($\vec{\phi}$). Entries of this vector correspond to loops in the nest from outermost to innermost, where each entry of this vector can be 1 or 0. An entry of 1 indicates that the corresponding loop is parallelised; otherwise, it is not parallelised. For example, for a nest with three loops, $\vec{\phi} = (1 \quad 1 \quad 0)T$ indicates that the first two loops are parallelised, whereas the iterations of the inner loop are to be executed sequentially. While there are different ways of obtaining the $\vec{\phi}$ vector for each loop, we employ automatic loop parallelisation theory (Banerjee 1994) and identify the loops whose iterations could be executed in parallel. Alternately, one can allow the programmer to explicitly mark the parallel loops, as is also common in the high-end parallel computing community (Koelbel, Loveman and Schreiber 1993; The openmp application program interface, http://www.openmp.org/). This programmer-assisted parallelisation is typically supported with directives/pragmas, enabling users to actively assist the compiler in the parallelisation process.

A loop space contains all the iterations executed by the loop and is represented by a set whose boundaries are defined by loop limits. Consider, for example, the loop

nest given in Figure 4. We can represent the iteration space of this loop nest using the following Presburger formulation.

$$\mathcal{L} = \{(i_1, i_2, i_3) \mid L_1 \leq i_1 \leq U_1 \ \wedge \ L_2 \leq i_2 \leq U_2 \ \wedge \ L_3 \leq i_3 \leq U_3\}.$$

The set of loop iterations that will be executed by a processor is determined by parallelism vector (ϕ) as well as the scheduling strategy adopted. In this article, we employ a *static block scheduling*, where each processor is assigned a set of successive loop iterations. Assuming that we have $\vec{p} = (1\ 0\ 0)^T$ for the loop nest in Figure 4 and that we have four processors (PROC0 through PROC3), the set of loop iterations that will be executed by processor c (where $0 \leq c \leq 3$) can be determined as:

$$\mathcal{K}(c) = \{(k_1, k_2, k_3) \mid (L_1 + c(U_1 - L1 + 1)/4 \leq k_1 < L1 + (c+1)(U_1 - L1 + 1)/4 - 1) \ \wedge \ (L_2 \leq k_2 \leq U_2) \ \wedge \ (L_3 \leq k_3 \leq U_3)\}.$$

Our approach, which is described in the rest of this article, is independent of the parallelisation strategy employed.

We perform the data distribution, which is followed by code instrumentation. We then execute the instrumented code to extract the profiling information. This way we obtain the communication pattern for each loop nest in the application. The communication calls are then inserted automatically based on the information provided by the profiler. We also need to mention that, in deciding data distribution, we tried to place the data elements frequently used by a processor into the local memory of the node which holds that processor (i.e. we tried to improve access locality to reduce communications as much as possible). After this step, the loops in the application are parallelised based on the data distribution. As a result of this parallelised data and computation distribution, we generate an affinity table to show the pairwise interprocessor communication among the nodes of the NoC-based CMP. While this part of the system has not been completely automated yet, we are able to generate close estimates and use them in our implementation for preliminary tests.

Execution of the program is viewed as a series of *phases*, where each phase corresponds to the communications taking place within a loop. Processors get synchronised before they start executing the next phase. Once the interprocessor communication requirements are identified, the communication pattern of the entire program can be represented using a *table* on which our approach operates. In the rest of the article, this table is called the *affinity table*. As shown in Figure 2, this information is then fed to the specific tool used, mainly, our GA-based implementation or the ILP-based implementation. The goal of our GA implementation and ILP formulation, explained in the next two sections, is to distribute the nodes of the NoC-based CMP to minimise the overall communication and computation throughout the communication phases.

```
for (i1 = L1; i1 ≤ U1; i1 ++)
for (i2 = L2; i2 ≤ U2; i2 ++)
  for (i3 = L3; i3 ≤ U3; i3 ++)
    ··· = A[i2, i3 + i1] + A[i2 + i3, i3 − i2] + A[i2 + 2, i3 + i1 − 1];
```

Figure 4. An example loop nest.

Note that, our approach can be extended to multi-application scenarios by generating the affinity table entries for every task pair. This information can be used to decide on selecting the nodes for such multi-application environments. However, problem will get more and more complicated with the increased number of concurrent applications as there will be much more constraints and cost factors.

4. Evolutionary framework

4.1. Preliminaries

Evolutionary computation provides strong solutions to multiple input parameters, which makes it a suitable candidate for computer-aided design (CAD) implementations. Evolutionary computational techniques start with an *initial population*, where individuals of the population are generated through variations in the input parameters. New generations are generated using new models created through *cross-over* and *mutation*. Similar to natural *selection*, some of the individuals in the population are promoted to the next generation using a *fitness function*. Cross-over is performed on the highest fitness individuals, thereby forcing low-fitness individuals disappear.

4.2. Algorithm

As explained before, we try to minimise the communication and computation of a given parallel application by selection and placement of nodes through a GA. We start our GA implementation with a set of randomly generated chromosomes which form our initial population. The size of this initial population is selected based on the NoC size and the number of different type of processors, as it greatly affects the solution time. In our GA implementation, chromosomes are represented as triplets, (t, x, y), where t indicates the type of processor used for that node, whereas (x, y) indicates the coordinates of the location of the node. In Figure 5a, a heterogeneous NoC with four nodes is shown. The corresponding chromosome sequence for this NoC is given as (1,0,5) → (2,1,5) → (2,3,5) → (3,0,3). The initial population is generated using these chromosome sequences. Algorithm 1 recursively generates the initial population with n chromosomes. Each recursive call tries to randomly select a CPU type for one of the nodes(b) in the set of N nodes. Then it tries to place this selected node in the chip area, which is performed by the $(x, y) = place(b,C_X,C_Y,$ *placement*) call. Thus, a chromosome (t, x, y) is assigned to every node in the initial N node set. Every valid chromosome created is appended to the *placement* set. This recursive call is performed for each individual to be created in the initial population.

Table 1 gives the constant terms used in our implementation. These constant terms are used for both GA and ILP. Assume that we are given N number of nodes, where $1 \leq i \leq N$. Our algorithm places these nodes on the 2D grid with (C_X, C_Y) dimensions, and at the end, returns the coordinates of each NoC-based CMP node. Communication load between two nodes is expressed by $A_{i,j}$, which indicates the affinity between two nodes as explained in Section 3.2.

The most well-known genetic operators to generate offsprings are mutation and cross-over. A certain proportion of the population is selected to generate a new population using a fitness function. Specific fitness function we employ is given as

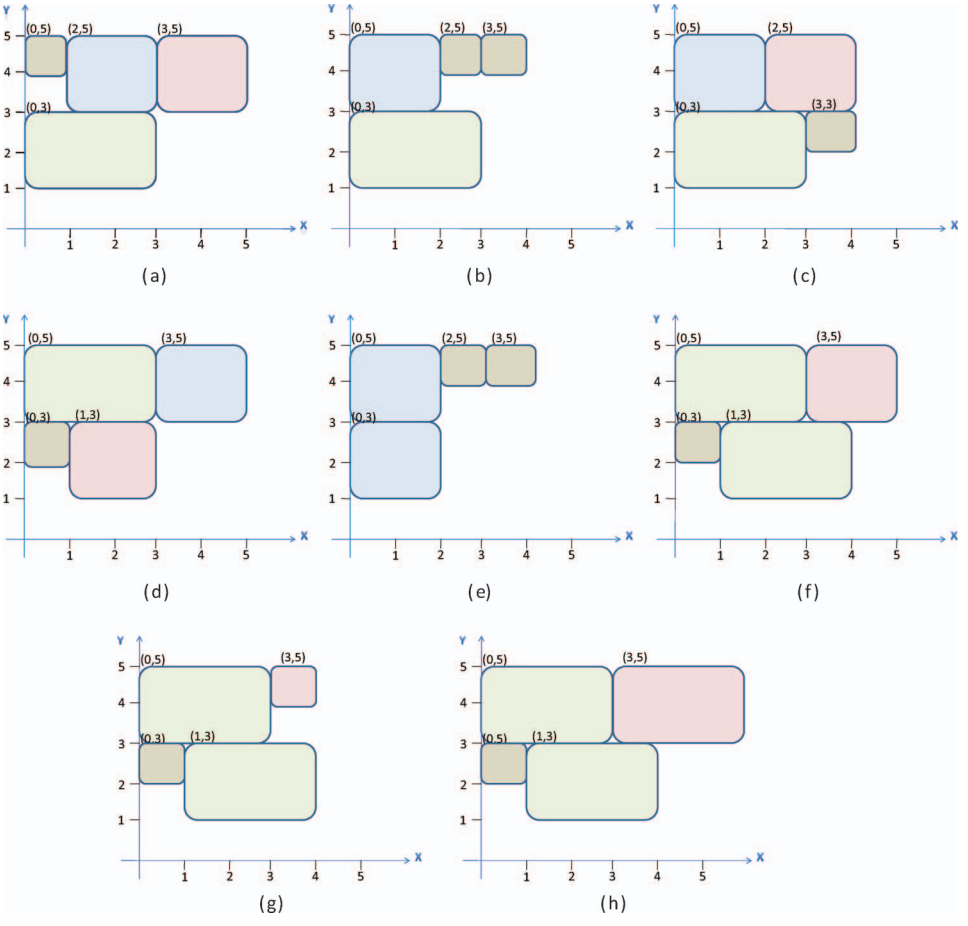

Figure 5. (a-d) Initial population starts with four individuals generated through our algorithm ($G1$, $G2$, $G3$, $G4$). (e-f) Two offspring ($G5$ and $G6$) generated by a cross-over on $G2$ and $G4$. (g) A legal mutation on $G6$. (h) An illegal mutation on $G6$.

Table 1. The constant terms used in our GA/ILP implementation. These are either architecture specific or program specific. $A_{i,j}$ values are obtained as explained in Section 3.2.

Constant	Definition
N	Number of nodes
P	Number of processor types
C_X,C_Y	Dimensions of the chip
X_i,Y_i	Dimensions of processor type i
$A_{i,j}$	Affinity between nodes i and j
L_i	Computation load in node i
Cap_i	Processing capacity of processor type i
CUC	Communication unit cost
PUC	Processing unit cost

Algorithm 1: Algorithm to generate a random individual with n chromosomes recursively.

```
function generatePopulation(C_X,C_Y,placement,N)
  if b ! = null then
    b ⇐ random CPU type t
    (x, y) = place(b,C_X,C_Y,placement)
    if (x > 0 and y > 0) then
      add chromosome (t,x,y)
      generatePopulation(C_X,C_Y,placement∪ (t,x,y),N–1)
    else
      generatePopulation(C_X,C_Y,placement,N)
    end if
  end if
```

total communication and computation cost of the design. This is evaluated using the following fitness rate function.

$$\sum_{i=1}^{N}(L_i/Cap_i) \times PUC + \sum_{i=1}^{N}\sum_{j=1}^{N} Distance_{i,j} \times A_{i,j} \times CUC. \text{ s.t. } i \neq j. \quad (1)$$

Note that in the above formulation PUC and CUC indicate the processing unit cost and communication unit cost, respectively. As can be seen in Algorithm 2, the communication cost between every pair of nodes is given with $Distance_{i,j} \times A_{i,j}$ (affinity). The computation cost of each node is also added to the overall design cost. The highest fitness individuals are selected to generate the next generation as this increases the probability of higher fitness individuals. While roulette wheel selection and tournament selection techniques are the most common techniques, we use a wheel roulette technique to select two optimum chromosome sequences (design patterns). Every individual in the population is evaluated based on the fitness rate and two best individuals are used to generate the new generation chromosome sequences. Figure 5a–d shows four individuals, namely $G1$, $G2$, $G3$ and $G4$, with different chromosome sequences. Table 3 shows their affinities and respective loads. The last column of the table indicates the load on each node, whereas the first four columns show the affinities with other nodes. Note that affinity between nodes i and j can be seen on the ith row and jth column. Similarly, Table 2 shows the example processor types and their respective properties. Last column of this table indicates the processing capacity of each processor, Cap_i. Based on these values, fitness rates for $G1$, $G2$, $G3$ and $G4$, are 84.2, 79.5, 84.3 and 79.9, respectively. Note that these are obtained using the fitness function given in Expression 1, processor properties given in Table 2, and the affinities given in Table 3. Among these randomly generated individuals, the

Table 2. Processor types and their respective properties.

Processor	Dimensions	Capacity
1	1 × 1	10
2	2 × 2	30
3	2 × 3	50

Table 3. Example processor types and their respective properties. The last column indicates the load on each node, whereas the first four columns show the affinities with other nodes.

Node	1	2	3	4	Load
1	0	3	2	1	40
2	3	0	5	4	60
3	2	5	0	8	100
4	1	4	8	0	80

Algorithm 2: Algorithm to calculate fitness rate of each chromosome sequence in terms of total cost.

```
function calculateFitness(selectedCPUs[N])
  for i ← 1 ... N do
    for j ← i ... N do
      commCost ⇐ commCost + Distance_{i,j} × A_{i,j}
    end for
    compCost ⇐ compCost + L_i/Cap_i
  end for
  totalCost ⇐ compCost + commCost
```

two chromosome sequences with the best fitness rates are $G2$ (b) and $G4$ (d). Therefore we select these two individuals to generate the next generation.

The parent selection process is followed by a cross-over or reproduction to form offsprings. In cross-over, two parents exchange their genome parts. This change is performed at a cross-over point which can be determined randomly or reasonably. By recombining partial solutions (chromosome sequence parts), cross-over enables GA to reach promising regions of the search space quickly. In our implementation, the cross-over point is selected to be the half of the chromosome sequence. As shown in Figure 5e and f, two new offsprings are generated by using the chromosome sequences $G2$ and $G4$ (the individuals with best fitness rates). Chromosome sequences for parents are (2,0,5) → (1,2,5) → (1,3,5) → (3,0,3), and (3,0,5) → (2,3,5) → (1,0,3) → (2,1,3). Therefore, generated offsprings $G5$ and $G6$ have (2,0,5) → (1,2,5) → (1,4,5) → (2,0,3) and (3,0,5) → (2,3,5) → (1,0,3) → (3,1,3) sequences since we used 2 as our cross-over point. Each offspring is checked for legality, since the newly generated offspring formed by combining the two halves of the parent chromosomes may not fit onto the available chip area. In our example, the new population after performing cross-over becomes ($G1$, $G2$, $G3$, $G4$, $G5$, $G6$). Algorithm 3 shows how our cross-over operates on the genome parts.

As can be seen in the last part of Algorithm 3, we keep the population size constant by removing the low-fitness individuals. This type of GA is called Steady-State GA, where a survival function replaces a set of individuals from the old population with the new generation. In our specific example, we evaluate chromosome $G1$ through $G6$ and select the four highest-fitness individuals as the next generation. More specifically, when new members of the population, $G5$ and $G6$, are considered we see that their fitness rates are 95 and 77.8, respectively. Therefore $G6$ replaces $G3$ in the new generation, whereas $G5$ disappears due to its low fitness rate. Consequently, our new population becomes ($G1$, $G2$, $G4$, $G6$). In a sense, we try to mimic natural selection through the fitness function.

Algorithm 3: Cross-over algorithm to form new offsprings.

```
function cross - over(Max1,Max2,population)
  for i ← 1 ... N/2 do
    Child1 ⇐ Max1[i]
    Child2 ⇐ Max2[i]
  end for
  for i ← N/2 + 1 ... N do
    Child1 ⇐ Max2[i]
    Child2 ⇐ Max1[i]
  end for
  if (Child1 is legal) then
    population = population + child1
    new + +
  end if
  if (Child2 is legal) then
    population = population + child2
    new + +
  end if
  if (new > 0) then
    population ⇐ population - getMin(population)
  else
    if (new > 1) then
      population ⇐ population - getMin(population)
      population ⇐ population - getMin(population)
    end if
  end if
```


Mutation is a key operator in biological evolution, as it prevents convergence to local optima by sampling new points in the search space. In our GA implementation, we implement mutation by selecting a random chromosome, that is, we modify the processor type of a randomly selected node. Note that, this newly generated chromosome may not be legal, as it may not fit onto the NoC. Our algorithm applies another random mutation if this change is not legal. For example, Figure 5g and h shows two mutations on $G6 = (3,0,5) \rightarrow (2,3,5) \rightarrow (1,0,3) \rightarrow (3,1,3)$. Both of these mutations are applied on the second chromosome – (2,3,5) –, creating $(3,0,5) \rightarrow (1,3,5) \rightarrow (1,0,3) \rightarrow (3,1,3)$ and $(3,0,5) \rightarrow (3,3,5) \rightarrow (1,0,3) \rightarrow (3,1,3)$. The former is legal since it becomes a smaller processor, a type 1 processor instead of type 2, whereas the latter is not legal due to insufficient chip area. Algorithm 4 shows how our mutation approach works on a chromosome.

A population is generated through many stages of reproduction until a termination condition has been reached. The termination condition can be a fixed number of iterations, an acceptable level of fitness rate, allocated budget, time or cost. On top of these conditions, we check whether the cross-over operation is generating new offspring or not. This enables faster search space exploration. Recall that, for our running example, the population before the mutation is ($G1$, $G2$, $G4$, $G6$). According to this population and the fitness function, $G2$ and $G6$ are selected as the parents to form the next generation. Cross-over generates two offspring, namely, $G7 = (2,0,5) \rightarrow (1,2,5) \rightarrow (1,3,5) \rightarrow (3,0,3)$ and $G8 = (3,0,5) \rightarrow (2,3,5) \rightarrow (1,0,3) \rightarrow (3,1,3)$. Note that these two offspring are same as their parents except with a different orientation. In our implementation, we terminate the GA if the offsprings are same as their parents (even with different orientations).

Algorithm 4: Mutation on a chromosome.

```
function mutation(placement)
  b ⇐ random CPU type t
  (x, y) = re - place(b,C_X,C_Y,placement - b)
  if (x > 0 and y > 0 then
    add chromosome (t,x,y)
    placement ⇐ placement - b ∪ (t,x,y)
  else
    mutation(placement)
  end if
```

4.3. Implementation and discussion

Our initial experimental evaluation shows that the GA-based approach is both practical and promising. It is practical because we are able to achieve close-to-optimal results within the fractions of the time ILP executes. More specifically, in our benchmarks, we generate results between 25 s and 3 min. Our proposed solution is promising as the initial results collected so far indicate significant savings.

Note that, in our GA implementation, we employ area, performance and energy as constraints, whereas temperature and communication bandwidth and other possible constraints are left out. For example, depending on the switch present in a node, bandwidth available to the connected links will be limited. Our implementation, in its current form, does not cover this constraint (and similar ones). However, our GA-based implementation can easily be modified to include such constraints.

Note also that, we do not consider possible energy reduction within the communication channels which could be a possible extension to this problem. As some of the links might be unused most of the time, it can be possible to shut these links down to save energy. In our future studies, we would like to explore the aforementioned objectives using a multi-objective GA framework.

5. ILP formulation

Our goal in this section is to present an ILP formulation of the problem of minimising communication and computation by determining the optimal selection and placement of nodes in an NoC-based CMP. This ILP framework is used to compare with our GA-based implementation presented in the previous section.

ILP provides a set of techniques that solve those optimisation problems in which both the objective function and constraints are linear functions and the solution variables are restricted to be integers. The 0–1 ILP is an ILP problem in which each (solution) variable is restricted to be either 0 or 1 (Nemhauser and Wolsey 1988). As indicated before, Table 1 gives the constant terms used in our ILP/GA formulation. We used *Xpress-MP* (2002), a commercial tool, to formulate and solve our ILP problem, though its choice is orthogonal to the focus of this article. In our ILP formulation, we view the chip area as a 2D grid, and assign nodes into this grid.

As before, we assume that we are given N number of nodes with dimensions (X_i, Y_i). Our ILP approach uses 0–1 variables to place these nodes on the 2D grid with (C_X, C_Y) dimensions, and at the end, returns the coordinates of each NoC-based CMP node.

In order to express the location of node n, we use the NC variable. More specifically,

- $NC_{n,x,y}$: indicates whether node n is in (x, y).

The mapping between the coordinates and nodes is ensured by the Map variable. That is,

- $Map_{x,y,n}$: indicates whether coordinate (x,y) is assigned to node n.

We capture the distance between two nodes by using the distances on both x and y axis, where $\Delta X_{i,j,x}$ returns the distance on x-axis and $\Delta Y_{i,j,x}$ returns the distance on y-axis. Specifically, we have:

- $\Delta X_{i,j,x}$: indicates whether the distance between nodes i and j is equal to x on the x-axis.
- $\Delta Y_{i,j,y}$: indicates whether the distance between nodes i and j is equal to y on the y-axis.

The processor type being selected is identified with:

- $PT_{n,p}$: indicates whether node n is running processor type p.

After describing the 0–1 variables, we next, give our constraints. Our first constraint is regarding the unique assignment of a node, that is, a node can be assigned to a single coordinate on the grid:

$$\sum_{i=1}^{C_X}\sum_{j=1}^{C_Y} NC_{n,i,j} = 1, \quad \forall n. \tag{2}$$

In the above equation, i and j correspond to the x and y coordinates, respectively. Each node must be assigned only one type of processor:

$$\sum_{j=1}^{P} PT_{n,j} = 1, \quad \forall n. \tag{3}$$

We make sure that each node is assigned only one type of processor by setting the overall sum to 1, which will force the ILP solver to set only one of the 0–1 variables to 1. Manhattan distance is assumed to be the unit cost of the *data communication* between two nodes, and is the metric whose value we use in minimisation. To capture the Manhattan distance, we use two variables, namely, $\Delta X_{i,j,x}$ and $\Delta Y_{i,j,y}$, and employ the following constraints:

$$\begin{aligned} &\Delta X_{n_1,n_2,x} \geq NC_{n_1,x_1,y_1} + NC_{n_2,x_2,y_2} - 1, \\ &\forall n_1, n_2, x, x_1, x_2, y_1, y_2, \text{ s.t.} \\ &\quad x = |x_1 - x_2| \text{ and } n_1 \neq n_2. \end{aligned} \tag{4}$$

$$\begin{aligned} &\Delta Y_{n_1,n_2,y} \geq NC_{n_1,x_1,y_1} + NC_{n_2,x_2,y_2} - 1, \\ &\forall n_1, n_2, y, x_1, x_2, y_1, y_2, \text{ s.t.} \\ &\quad y = |y_1 - y_2| \text{ and } n_1 \neq n_2. \end{aligned} \tag{5}$$

Having specified the necessary constraints in our ILP formulation, we next give our objective function. We define our total cost function as the sum of the communication cost in both dimensions and computation cost. More specifically, we denote the total data communication using $Comm_X$ and $Comm_Y$ for dimensions x and y, respectively:

$$Comm_X = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{C_X} A_{i,j} \times \Delta X_{i,j,k} \times k \text{ s.t. } i \neq j. \tag{6}$$

$$Comm_Y = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{C_Y} A_{i,j} \times \Delta Y_{i,j,k} \times k \text{ s.t. } i \neq j. \tag{7}$$

Note that, for both communication costs, we use $A_{i,j}$ to express the affinity between two nodes. This is multiplied with the distance given by k which is also multiplied with 0–1 variable $\Delta X_{i,j,k}$ or $\Delta Y_{i,j,k}$. This variable indicates whether k is the actual distance between nodes i and j. Consequently, our communication cost can be expressed as:

$$Comm = (Comm_X + Comm_Y) \times CUC. \tag{8}$$

Next, we give the computation cost as:

$$Comp = \sum_{i=1}^{N} \sum_{j=1}^{P} PT_{i,j} \times L_i \times PUC. \tag{9}$$

To summarise, our node placement and processor selection problem can be formulated as 'minimise $Comm + Comp$ under constraints (2) through (9)'. It is important to note that this ILP formulation is very flexible as it can accommodate different numbers of nodes and dimensions. Note also that the ILP formulation has many other constraints which are not shown for clarity reasons.

6. Experimental evaluation

6.1. Setup

We performed experiments with eight different array-based benchmark codes parallelised through an optimising compiler built upon SUIF. Table 4 lists the applications and their important characteristics. In collecting these results, we fast-forwarded the first 1 billion instructions, and simulated the next 300 million instructions. The fourth column of Table 4 gives the number of data accesses for each application. While GA solution times varied between 25 s and 3 min, averaging on about 94 s, ILP solution times were between 172 min and 18.4 h. In our base configuration, we used four different types of processors with properties given in Table 5.

We performed experiments with four different execution models for each benchmark code in our experimental suite:

- *Homogeneous:* This is in a sense a conventional NoC topology which uses the same type of processors in the NoC nodes.

Table 4. Benchmark codes used in this study.

Benchmark	Source	Description	Number of data accesses
3step-log	DSPstone	Motion estimation	90646252
adi	Livermore	Alternate direction integration	71021085
ammp	Spec	Computational chemistry	86967895
equake	Spec	Seismic wave propagation simulation	83758249
mcf	Spec	Combinatorial optimisation	114662229
mesa	Spec	3D graphics library	134791940
vortex	Spec	Object-oriented database	163495955
vpr	Spec	FPGA circuit placement	117239027

Table 5. Processors used in our heterogeneous NoC and their characteristics.

Processor	IPC	Energy	Area
P_1	1	3.73	1
P_2	1.3	6.88	2
P_3	1.87	10.68	8
P_4	2.14	46.44	40

- *Random:* This is a randomly generated heterogeneous NoC, where the processor types are selected randomly and placement of these nodes are done randomly.
- *GA:* This is the GA-based processor selection and node placement strategy for heterogeneous NoCs, wherein a close-to-optimal placement is targeted.
- *ILP:* This is the ILP-based processor selection and node placement strategy, which we compare our GA-based implementation with. This scheme represents the optimal placement for heterogeneous NoCs.

6.2. Results

Figure 6 gives our total cost results normalised with respect to the homogeneous scheme based on four processors. We see that the overall average normalised costs with random and GA are around 110% and 77%, respectively. On the other hand, the ILP scheme has a normalised cost of 71% on average. Our GA-based implementation achieves close to optimal results (77% vs. 71%) with much better solution times. Specifically, our GA solution time varied between 25 s and 3 min, averaging on about 94 s, whereas ILP solution times were between 172 min and 18.4 h.

One can see that, while some of the benchmarks prefer a homogeneous NoC topology, as in the case of *ammp* and *vortex*, most of the benchmarks take advantage of heterogeneous NoC. For *ammp* and *vortex*, the optimum topology returned by the ILP is a homogeneous NoC with a 100% normalised total cost with respect to homogeneous. While heterogeneity provides more options and reduces the total cost, optimal placement also plays a key role in achieving the best results. In other words, the best design is achieved by combining both optimal placement and heterogeneous properties.

Recall that the original number of processor types we used were four. The bar-chart in Figure 7 shows the normalised costs (with respect to those of the Homogeneous scheme) for the benchmark equake with different numbers of processor types (the results with the original number of processors are also shown for convenience), ranging from 1 to 4. Note that, the total chip area is kept constant for all these experiments and the only difference between two experiments is the number of processor types and corresponding properties such as energy, area and performance. Available processor types and their characteristics are listed in Table 5. The second column gives the IPC value of

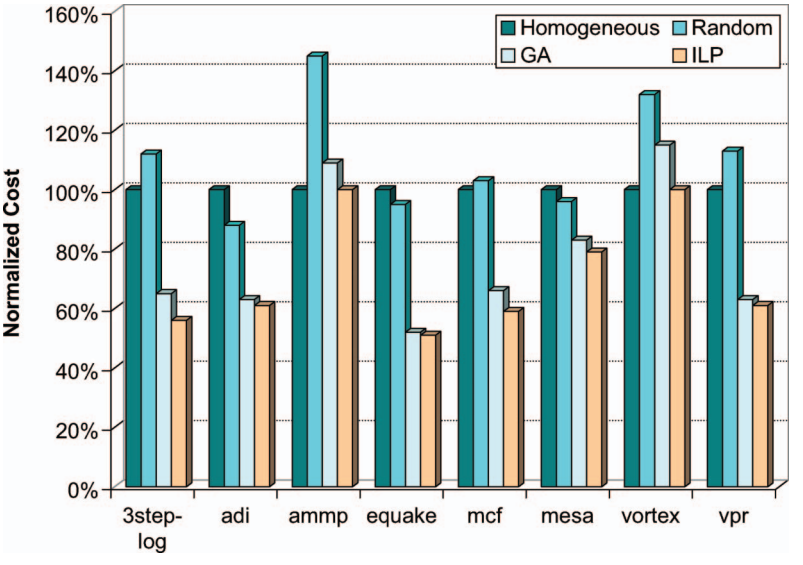

Figure 6. Total costs of random, GA and ILP normalised with respect to homogeneous.

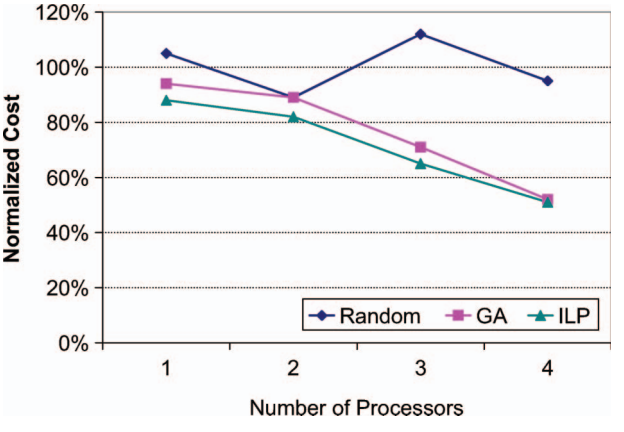

Figure 7. Normalised costs with the different number of processor types (equake).

each processor type, while the third column shows the average energy consumption and the last column shows the area required for the processor. We see from these results that the effectiveness of the GA-based and ILP approaches increases with the increasing number of processor types. The main reason for this behaviour is that adding more processor types gives more flexibility to select a better match for a given processing load. Note that, when only one type of processor is used, all the schemes behave similar to homogeneous except the placement of different nodes. GA and ILP try to optimise the location of different nodes depending on their affinities. This shows that optimal placement should be targeted along with heterogeneity.

In the next set of experiments, we include an additional execution model to reflect the behaviour of the studies (Ascia et al. 2004; Zhou et al. 2006; da Silva et al. 2009) described in Section 2. Recall that, these studies try to map IPs or processors onto the NoC using a GA-based framework with different objectives.

Random + GA: This is a randomly generated heterogeneous NoC, where the processor types are selected randomly, however placement is done using our GA-based implementation.

Note that, GA portion of this execution model is different from the previously proposed techniques (Ascia et al. 2004; Zhou et al. 2006; da Silva et al. 2009). However, it still shows the affects of separating processor selection from mapping. Figure 8 gives results for both *Random + GA* and *GA* normalised with respect to *Homogeneous* scheme. Average normalised costs with *Random + GA* and *GA* are around 94% and 77%, respectively. These results clearly show that processor selection plays an important role in achieving good results.

In our last set of experiments, we measure the ratio of communication to computation in GA normalised with respect to homogeneous. The bar-chart in Figure 9 shows the normalised costs for the communication and computation in the total cost. Recall that the original communication and computation formulation is used in both GA and ILP. As we can see from this graph, certain applications are more communication oriented, whereas others are computationally intense.

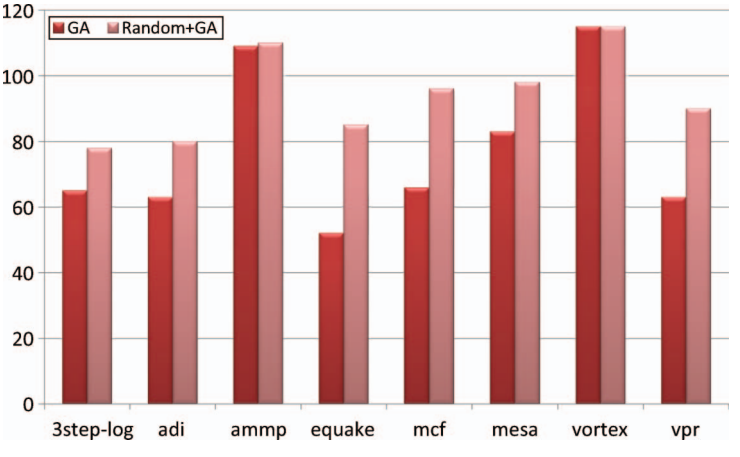

Figure 8. Normalized costs with *Random + GA*.

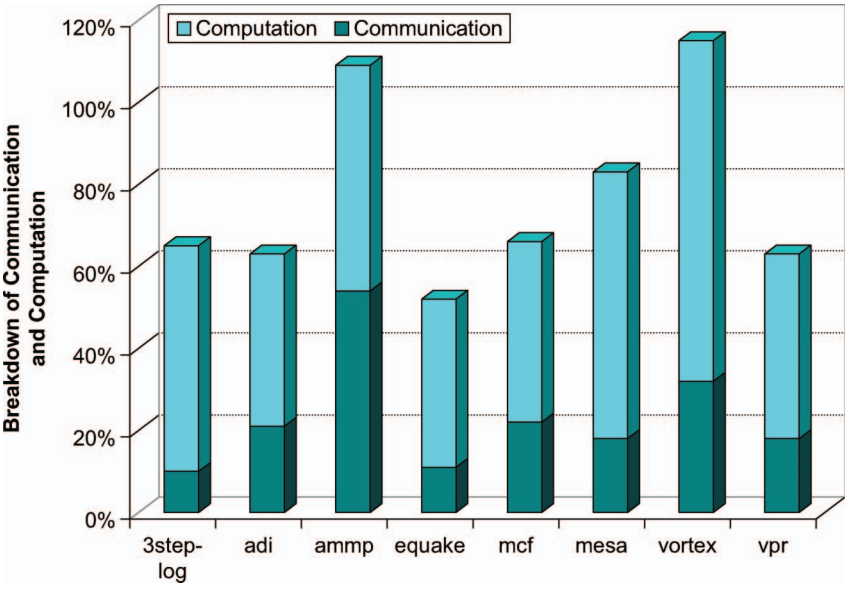

Figure 9. Communication vs. computation cost normalised with respect to homogeneous.

7. Concluding remarks

One of the promising solutions for designing efficient NoC-based CMPs is to make use of heterogeneity as it allows the processor to better match execution resources. The work described in this article studies application-specific placement of nodes in an NoC-based heterogeneous CMP. We expressed this problem using an evolutionary computation-based approach. Our preliminary experiments show that our approach is both practical and promising.

Acknowledgements

This research is supported in part by TUBITAK grant 108E233, by a grant from IBM, and by a Marie Curie International Reintegration Grant within the 7th European Community Framework Programme.

References

Ahonen, T., and Nurmi, J.A. (2007), 'Hierarchically Heterogeneous Network-on-Chip', in *Proceedings of EUROCON*, Warsaw: The International Conference on "Computer as a Tool", pp. 2580–2586.

Ascia, G., Catania, V., and Palesi, M. (2004), 'Multi-objective Mapping for Mesh-based NoC Architectures', in *Proceedings of the 2nd IEEE/ACM/IFIP international Conference on Hardware/Software Codesign and System Synthesis,* Stockholm, Sweden, September 08–10, 2004. CODES + ISSS '04, New York, NY: ACM, pp. 182–187.

Ascia, G., Catania, V., Palesi, M., and Parlato, A. (2003), 'An Evolutionary Approach for Reducing the Energy in Address Buses', in *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*, pp. 76–81.

Balakrishnan, S., Rajwar, R., Upton, M., and Lai, K. (2005), 'The Impact of Performance Asymmetry in Emerging Multicore Architectures', in *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pp. 506–517.

Banerjee, U. (1994), *Loop Parallelization*, Norwell, MA, USA: Kluwer Academic Publishers.

Benini, L., and De Micheli, G. (2001), 'Powering Networks on Chips: Energy-efficient and Reliable Interconnect Design for SoCs', in *Proceedings of ISSS*, Montreal, Canada: ACM, pp. 33–38.

Blume, H., Feldkaemper, H.T., and Noll, T.G. (2005), 'Model-based Exploration of the Design Space for Heterogeneous Systems on Chip', *Journal of VLSI Signal Processing Systems*, 40, 19–34.

Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A. (2004), 'Qnoc: QoS Architecture and Design Process for Network on Chip', *Journal of Systems Architecture*, 50, 105–128.

Brisolara, L., il Han, S., Guerin, X., Carro, L., Reis, R., Chae, S.I., and Jerraya, A. (2007), 'Reducing Fine-grain Communication Overhead in Multithread Code Generation for Heterogeneous mpsoc', in *SCOPES '07: Proceedings of the 10th international workshop on Software & compilers for embedded systems,* New York, NY, USA: ACM, pp. 81–89.

Cardoso, R.S., Kreutz, M.E., Carro, L., and Susin, A.A. (2005), 'Design Space Eexploration On Heterogeneous Network-On-Chip', in *Proceedings IEEE International Symposium on Circuits and Systems*. Washington, DC, USA: IEEE, pp. 428–431.

da Silva, M.V.C., Nedjah, N., and deMacedoMourelle, L. (2009), 'Application Synthesis for MPSoCs Implementation using Multiobjective Optimization', *Lecture Notes in Computer Science*, 5517, 736–743.

Dally, W.J., and Towles, B. (2001), 'Route Packets, not Wires: On-chip Interconnection Networks', in *Proceedings of DAC*, Las Vegas, Nevada: ACM, 684–689.

Eisley, N., and Peh, L.-S. (2004), 'High-level Power Analysis of on-chip Networks', in *Proceedings of CASES*, Washington, DC: ACM, pp. 104–115.

Flachs, B., Asano, S., Dhong, S., Hotstee, P., Gervais, G., Kim, R., Le, T., Liu, P., Leenstra, J., Liberty, J., Michael, B., Oh, H., Mueller, S., Takahashi, O., Hatakeyama, A., Watanabe, Y., and Yano, N. (2005), 'A Streaming Processing Unit for a Cell Processor (Vol. 1)', in *2005 IEEE International Conference on Solid-State Circuits, Digest of Technical Papers*. Washington, DC, USA: IEEE, pp. 134–135.

Gerstlauer, A. (2003), 'Communication Abstractions for System-level Design and Synthesis', Technical Report. TR-03-30, Center for Embedded Computer Systems, University of California, Irvine, CA.

Ghiasi, S., Keller, T., and Rawson, F. (2005), 'Scheduling for Heterogeneous Processors in Server Systems', in *CF '05: Proceedings of the 2nd conference on Computing frontiers*, New York, NY, USA: ACM, pp. 199–210.

Handl, J., and Knowles, J.D. (2007), 'An Evolutionary Approach to Multiobjective Clustering', *IEEE Transactions on Evolutionary Computation*, 11, 56–76.

Jena, R.K., and Sharma, G.K. (2007), 'A Multi-objective Evolutionary algorithm- Based Optimisation Model for Network on Chip Synthesis', *International Journal of Innovative Computing and Applications*, 1, 121–127.

Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R., and Shippy, D. (2005), 'Introduction to the Cell Multiprocessor', *IBM Journal of Research and Development*, 49, 589–604.

Koelbel, C.H., Loveman, D.B., and Schreiber, R.S. (1993), *The High Performance Fortran Handbook*, Cambridge, MA, USA: MIT Press.

Kongetira, P., Aingaran, K., and Olukotun, K. (2005), 'Niagara: A 32-Way Multithreaded Sparc Processor', *IEEE Micro*, March/April, 21–29.

Kreutz, M., Marcon, C., Carro, L., Wagner, F., and Susin, A. (2005), 'Design Space Exploration Comparing Homogeneous And Heterogeneous Network-on-Chip Architectures', in *Proceedings of IEEE International of the 18th Annual Symposium on Integrated circuits and System Design*, Florianolpolis, Brazil: ACM, pp. 190–195.

Kumar, R., Tullsen, D.M., and Jouppi, N.P. (2006), 'Core Architecture Optimization for Heterogeneous Chip Multiprocessors', in *PACT '06: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Seattle, Washington, USA: ACM, pp. 23–32.

Kumar, R., Tullsen, D.M., Jouppi, N.P., and Ranganathan, P. (2005), 'Heterogeneous Chip Multiprocessors', *Computer*, 38, 32–38.

Kumar, R., Tullsen, D.M., Ranganathan, P., Jouppi, N.P., and Farkas, K.I. (2004), 'Single-is a Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance', in *ISCA '04: Proceedings of the 31st Annual International Symposium on Computer Architecture*, New York, NY, USA: ACM, p. 64.

Lambrechts, A., Raghavan, P., Leroy, A., Talavera, G., Vander, T., Jayapala, M., Catthoor, F., Verkest, D., Deconinck, G., Corporaal, H., Robert, F., and Carrabina, J. (2005), 'Power Breakdown Analysis for a Heterogeneous NoC Platform Running a Video Application', in *Proceedings of the 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors*, Washington, DC, USA: IEEE Computer Society, pp. 179–184.

Leary, G., Srinivasan, K., Mehta, K., and Chatha, K.S. (2009), 'Design of Network-on-chip Architectures with a Genetic Algorithm-based Technique', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17, 674–687.

Lei, T., and Kumar, S. (2003), 'A Two-step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture', in *DSD '03: Proceedings of the Euromicro Symposium on Digital Systems Design*, Los Alamitos, CA, USA: IEEE Computer Society, p. 180.

Lieverse, P., Wolf, P.V.D., Vissers, K., and Deprettere, E. (2001), 'A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems', *Journal of VLSI Signal Processing Systems*, 29, 197–207.

Liu, F., and Chaudhary, V. (2003), 'Extending openmp for Heterogeneous Chip Multiprocessors', in *Proceedings of the 2003 International Conference on Parallel Processing*, Kaohsiung, Taiwan: IEEE Computer Society, pp. 161–168.

Loghi, M., Angiolini, F., Bertozzi, D., Benini, L., and Zafalon, R. (2004), 'Analyzing on-chip Communication in a MPSOC Environment', in *Proceedings of the conference on Design, Automation and Test in Europe,* Washington, DC, USA: IEEE Computer Society, p. 20752.

Madsen, J., Mahadevan, S., Virk, K., and Gonzalez, M. (2003), *Real-Time Systems Symposium*, Cancun, Mexico: IEEE Computer Society, pp. 265–274.

Mazumder, P., and Rudnick, E.M. (1999), *Genetic Algorithms for VLSI Design, Layout & Test Automation*, Upper Saddle River, NJ, USA: Prentice Hall PTR.

McGowen, R. (2005), 'Adaptive Designs for Power and Thermal Optimization, in *Proceedings the 2005 IEEE/ACM International Conference on Computer-Aided Design,* San Jose, CA: IEEE Computer Society, pp. 118–121.

Nemhauser, G.L., and Wolsey, L.A. (1988), *Integer and Combinatorial Optimization*, New York, NY, USA: Wiley-Interscience.

Palesi, M., and Givargis, T. (2002), 'Multi-objective Design Space Exploration Using Genetic Algorithms', in *CODES '02: Proceedings of the Tenth International Symposium on Hardware/software Codesign*, pp. 67–72.

Pham, D., Asano, S., Bolliger, M., Day, M., Hofstee, H., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T., and Yazawa, K. (2005), 'The Design and Implementation of a First-generation Cell Processor (Vol. 1)', in *2005 IEEE International Conference on Solid-State Circuits, Digest of Technical Papers*, pp. 184–592.

Schnecke, V., and Vornberger, O. (1996), 'A Genetic Algorithm for vlsi Physical Design Automation', in *Proceedings of ACEDC '96*. University of Plymouth, U.K.

Shee, S.L., Erdos, A., and Parameswaran, S. (2006), 'Heterogeneous Multiprocessor Implementations for jpeg: A Case Study, in *CODES + ISSS '06: Proceedings of the 4th International Conference on Hardware/software Codesign and System Synthesis,* New York, NY, USA: ACM, pp. 217–222.

Srinivasan, K., and Chatha, K.S. (2005), 'Isis: A Genetic Algorithm Based Technique for Custom on-chip Interconnection Network Synthesis', in *International Conference on VLSI Design*, pp. 623–628.

Taylor, M.B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Johnson, P., Lee, J., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S., and Agarwal, A. (2002), 'The RAW Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs', *IEEE Micro*, 22(2), 25–35.

Wang, H., Zhu, X., Peh, L., and Malik, S. (2002), 'Orion: A Power-performance Simulator for Interconnection Networks', in *Proceedings of MICRO*, pp. 294–305.
Wang, Y., and Dang, C. (2007), 'An Evolutionary Algorithm for Global Optimization Based on Level-set Evolution and Latin Squares', *IEEE Transactions on Evolutionary Computation*, 11, 579–595.
Xpress-MP (2002). http://www.dashoptimization.com/pdf/Mosel1.pdf
Yan, J., and Zhang, W. (2007), 'Hybrid Multi-core Architecture for Boosting Single-threaded Performance', *SIGARCH Computer Architecture News*, 35, 141–148.
Zhou, W., Zhang, Y., and Mao, Z. (2006), 'Pareto based Multi-objective Mapping IP Cores onto NoC Architectures', in *IEEE Asia Pacific Conference on Circuits and Systems*, Singapore: IEEE, pp. 331–334.