Multi-Objective Mapping Optimization via Problem Decomposition for Many-Core Systems

Shin-Haeng Kang*, Hoeseok Yang†, Lars Schor†, Iuliana Bacivarov†, Soonhoi Ha*, and Lothar Thiele†

*School of Electrical Engineering and Computer Science
Seoul National University, 151-742 Seoul, South Korea
{shkang, sha}@iris.snu.ac.kr

†Computer Engineering and Networks Laboratory
ETH Zurich, CH-8092 Zurich, Switzerland
{firstname.lastname}@tik.ee.ethz.ch

***Abstract*—Due to the trend of many-core systems for dynamic multimedia applications, the problem size of mapping optimization gets bigger than ever making conventional meta-heuristics no longer effective. Thus, in this paper, we propose a problem decomposition approach for large scale optimization problems. We basically follow the *divide-and-conquer* concept, in which a large scale problem is divided into several sub-problems. To remove the inter-relationship between sub-problems, proper abstraction is applied. The divided sub-problems can be solved either in parallel or in a sequence. The mapping optimization problem on dynamic many-core systems is decomposed and solved separately considering the system state and architectural hierarchy. Experimental evaluations with several examples prove that the proposed technique outperforms the conventional meta-heuristics both in optimality and diversity of the optimized pareto curve.**

I. Introduction

Today's multimedia embedded systems face a continuously increasing demand in system performance, leading to the integration of several and possibly heterogeneous (many) cores on a single chip. While modern multimedia handheld devices as well as personal computers already have dual- or quad-core processors inside, the number of cores on a die keeps growing up to several dozens [14], [16], [18]. Moreover, the digital convergence trend is to have systems running multiple applications simultaneously in different combinations at different moments in time. Thus, the application software is no longer static or monolithic, but dynamic and concurrent [10], [17].

In this context, one of the key EDA problems is mapping optimization, i.e., how to assign tasks to cores, communication between tasks to the on-chip communication fabric, and how to arbitrate shared resources. Mapping optimization is known to be NP-hard, even for homogeneous multi-core systems [21], and several approaches have been proposed. One popular approach is the use of integer linear programming (ILP) [20], [22], that mathematically computes optimal solutions. However, due to the intractable time complexity, ILPs are not scaling well to large problem sizes. Another popular approach is using meta-heuristics, such as genetic algorithms (GA) [11], quantum-inspired evolutionary algorithms (QEA) [21], or ant colony optimizations (ACO) [8]. Meta-heuristics find near-optimal solutions in a significantly reduced time and provide pareto-optimal solutions, since typically multiple objectives are being investigated [19].

In many-core systems, however, these multi-objective meta-heuristics are no longer effective, due to the scale of the investigated problems. The main problem is convergency to local optima: the larger the design space is, the more likely is to converge to a local optimum. In this paper, we propose a multi-objective mapping optimization technique that overcomes this shortcoming. The proposed technique relies on problem decomposition into sub-problems that can be solved individually.

To briefly introduce the proposed technique, let us consider the mapping optimization example in Fig. 1. A task graph is shown in Fig. 1(a), in which vertices and directed edges denote concurrently running tasks and communication between them, respectively. Each task is annotated with utilization requirements. The link between processes 4 and 5 is particularly highlighted with a thicker arrow, denoting a highly communication intensive link. The target architecture is depicted in Fig. 1(b), illustrating a hierarchical system with two clusters connected via on-chip network. Each cluster includes two processors communicating via a fast intra-cluster communication medium *Cl*, that can be a shared bus or crossbar.[1] The utilization of a certain processor should not exceed 1.0 (i.e., computed as the sum of individual utilizations of tasks mapped to that processor), while the used communication bandwidth should be less than the system available bandwidth. With these utilization constraints respected, the objectives are to balance the workload over all processors in the system and to minimize the communication overhead at the same time. Since the intra-cluster communication media offers a larger bandwidth than the on-chip network in general, it is desirable to allocate communication intensive links inside a cluster, thus isolate it from inter-cluster communication.

First, let us show how traditional meta-heuristics struggle to obtain good solutions. Suppose that there is a population that implies the mapping candidate in Fig. 1(c) during optimization. This is not a good solution since it stresses the

[1]Note that in current many-core architectures the number of clusters and core per cluster is in fact much bigger.

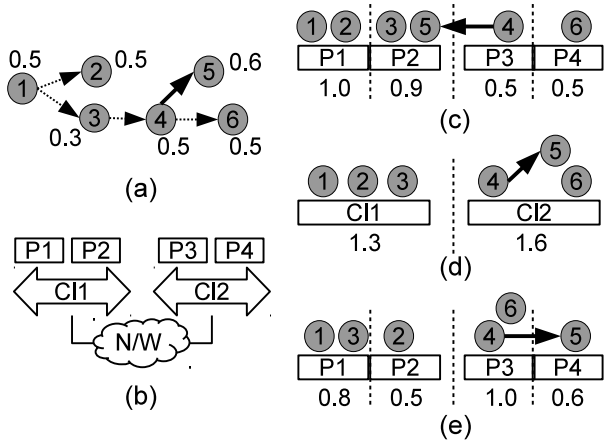

Fig. 1. Motivational example: (a) task graph annotated with required utilizations, (b) target hierarchical architecture, (c) an initial mapping, (d) task-to-cluster mapping variation from (c), and (e) refined task-to-processor mapping from (d).

on-chip network with intensive communication ($4 \rightarrow 5$).[2] Traditional meta-heuristics, however, may fail to remove this communication overhead with mapping variations, because there exists a tight utilization constraint which does not allow tasks 4 or 5 to be moved to the other cluster. To evolve to an efficient mapping like the one in Fig. 1(e), several changes should be made at the same time, which is usually avoided in meta-heuristics that try to preserve locality during optimization [15]. Generally speaking, the more constraints to be respected (or objectives to be optimized), the less freedom we are given during exploration.

Alternatively, we propose to divide the mapping problem into sub-problems, i.e., for this example, *task-to-cluster* and *task-to-processor* mappings, and solve them sequentially. In the first stage, only the *task-to-cluster* mapping is determined. At this level, the candidate shown in Fig. 1(c) can easily be evolved to Fig. 1(d) by assigning task 5 to the other cluster ($Cl2$), in which the intensive communication on the network is optimized. Then, in the second phase *task-to-processor* mapping can be further refined, resulting in the optimized mapping in Fig. 1(e). In other words, isolating some details in the first phase enables the exploration beyond the local optimum.

This decomposition can be seen as a hierarchical top-down optimization. That is, starting from the top level in the architecture hierarchy, the system is sequentially optimized and refined to lower levels. The intended positive effect is to reduce the possibility of convergence to a local minimum, as exemplified in Fig. 1. However, this comes at the cost of solution space reduction, i.e., compared to the conventional meta-heuristics that optimize over the entire design space, this decomposition might encounter losses for individual solution spaces of sub-problems, that may result in a sub-optimal overall solution. To compensate this loss, we propose to apply an additional conventional optimization at the end, with having the optimized solution as initial population, e.g., the solution of Fig. 1(e).

Three intuitive ideas lay behind this decomposition-based approach: *divide-and-conquer*, *prioritization*, and *abstraction*. By decomposing a problem into sub-problems, we take advantage of the reduced design space for each sub-problem, that is the principle behind *divide-and-conquer* procedures. These sub-problems are either independent or inter-related with each other. In case that they are independent, they can be optimized individually and, trivially, just merged in the end. However, tightly coupled sub-problems cannot be solved concurrently, but sequentially by successively refining the abstracted properties. The order in which sub-problems are handled is critical, since it affects the optimality of the solution. This should be decided by a proper *prioritization*. For instance, the on-chip network in Fig. 1 is critical compared to the intra-cluster communication with high bandwidth, and therefore *task-to-cluster* mapping is solved first, prior to *task-to-processor* decisions.

When dealing with tightly coupled sub-problems, their inter-relationships should be properly decoupled. In a sub-problem, missing details from the other sub-problems should still be handled, that is the *abstraction* principle. For instance, for *task-to-cluster* mapping, the processor load balancing should be evaluated without *task-to-processor* mapping information. To this end, we need to introduce *speculation methods* during the decomposition of tightly coupled sub-problems.

The main contribution of this paper is to propose a multi-objective mapping optimization technique for dynamic many-core systems with problem decompositions, in which three key principles, i.e., divide-and-conquer, prioritization, and abstraction, are implemented. With extensive case studies, we show that the mapping problem of dynamic application scenarios onto hierarchical many-core architecture is efficiently solved. The proposed decomposition-based multi-objective meta-heuristic framework could generally be applied to other large scale optimization problems.

II. Related Work

EDA optimization problems are typically defined by multiple objectives that have to be optimized simultaneously [2], [19], [24]. Therefore, solutions may not be comparable with each other, some being better in some of the objectives and worse in others. Thus, designers usually have to deal with pareto-optimal sets rather than single solutions. A simple approach is to aggregate all objectives into a single, parameterized objective function [3]. Though this approach is beneficial in the sense that well-established single-objective technique can be used seamlessly, it is sensitive to the shape of the pareto front, prior knowledge on the problem may be required, and it requires several optimization runs which do not exploit synergy from each other.

To overcome these shortcomings, multi-objective evolutionary algorithms (MOEA) have been proposed. To properly manage populations to evolve to a pareto-optimal set, non-dominated sorting [5], strength pareto evolutionary algorithm

[2]Note that all communications except the intensive one are omitted for brevity.

[26], and simple indicator algorithm [25] have been proposed. Problem specific specialization of operators (i.e., crossover and mutation) and different chromosomal representations have also been studied [3], [7], [9]. However, MOEAs are still not scaling well to large problems.

A decomposition-based MOEA has been introduced in [23], where the design space is decomposed in sub-spaces that can be individually optimized for a single objective. Neighborhood relations are defined among sub-spaces. While ordinary EAs are used to optimize these sub-spaces simultaneously, each still needs to consider the status of its neighbors. Though this technique might explore the design space faster, it is usually worse than ordinary MOEAs in terms of solution quality since it aggregate multiple objectives into one for individual optimizations. Moreover, as this work does not decompose the problem itself, the large problem size is still an issue.

Another decomposition approach for optimization problems has been proposed in [12] and formalized in [1]. The basic idea is that a set of valid and good sub-solutions compose a valid and good global solution, i.e., the *monotonicity* property. The main propositions justifying this decomposition technique are as follows: (1) the solution space of sub-problems is smaller than the global one, as the number of parameters to be optimized is smaller; (2) the evaluation effort for each sub-problem is lower; and (3) a larger number of system designs are evaluated in a small fraction of the whole solution space.

In our proposed technique, we hold the same propositions as above. However, we do not rely on the *monotonicity* property, which usually does not hold for complex problems such as mapping optimization. This is the key difference with our problem decomposition. The problem in [12] and [1] is assumed to be composed of sub-problems that are independent or loosely coupled in a monotonous relationship with each other. Our proposed technique, in contrast, is more generic, allowing sub-problems to have general inter-relationships. One of the key contributions of this paper is how to manage these relationships.

III. Problem Decomposition Framework

A multi-objective optimization problem (MOOP) is defined on the set of decision/objective spaces $\mathbf{X}/\mathbf{Y}$ (with x and y decision and objective vectors, respectively), and it optimizes a set of k objective functions f_i given the set of m constraints $e_i \in \mathbf{e}(\mathbf{x})$. The optimization problem can therefore be specified as follows:

$$\begin{array}{ll} \texttt{minimize} & \mathbf{y} = \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_k(\mathbf{x})) \\ \texttt{subject to} & \mathbf{e}(\mathbf{x}) = (e_1(\mathbf{x}), ..., e_m(\mathbf{x})) = (\text{true}, ..., \text{true}) \\ \texttt{where} & \mathbf{x} = (x_1, ..., x_n) \in \mathbf{X} = X_1 \times ... \times X_n \\ & \mathbf{y} = (y_1, ..., y_k) \in \mathbf{Y} = Y_1 \times ... \times Y_k \end{array} \quad (1)$$

Conventional EAs take *decision variables* and *constraints* as inputs and generate a set of optimized solutions. This is illustrated in the gray box on the right-top of Fig. 2, while our proposed technique uses several such EA basic blocks, as identified in the figure. The strategy is to decompose the multi-objective optimization problem ($MOOP$) into sub-problems

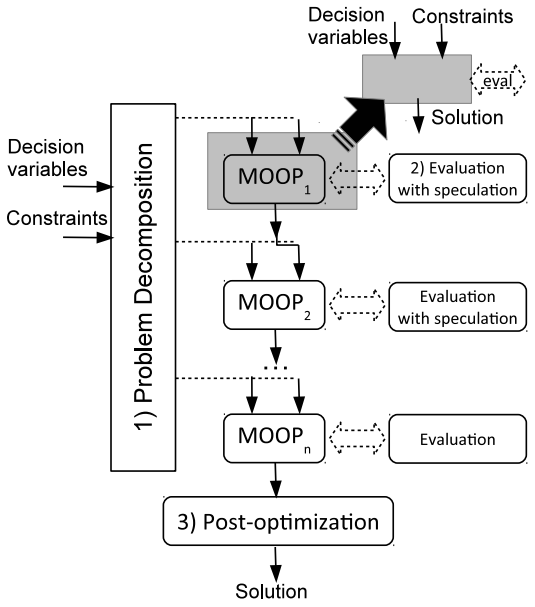

Fig. 2. Proposed problem decomposition framework.

($MOOP_i$) and then solve each sub-problem individually. Sub-problems can either be processed sequentially as illustrated in Fig. 2, or handled independently and in parallel if the problem allows it. In each sub-problem, mapping solutions have to be evaluated separately and independently of the design spaces of other sub-problems. The decomposition causes incomplete information in sub-problems, and evaluations can only be done via so-called *speculations*. Finally, to compensate the loss of the reduced solution space that is also a consequence of the decomposition (as defined in section I), a *post-optimization* procedure is executed. Note that the proposed decomposition can be done in a hierarchical manner, i.e., each $MOOP_i$ may have another decomposition layer inside.

To summarize, we address large scale problems as follows:

1) *Decomposition.* The given problem is decomposed into sub-problems, and updated sets of decision variables and constraints are defined for each sub-problem.
2) *Speculation.* If a sub-problem does not have complete information for evaluation of the design space, we derive an associated abstract method called *speculation*.
3) *Post-optimization.* To reach global optimality, a final global optimization is conducted having the derived solutions of the decomposition chain as initial population.

The following sub-sections are detailing each of above procedures.

A. Problem Decomposition

To express the dimension reduction of the decision space due to problem decomposition, we define following relation on two domains in the decision space $X^i = X_1^i \times X_2^i \times ... \times X_p^i$ and $X^j = X_1^j \times X_2^j \times ... \times X_q^j$ such that $q \leq p$,

$X^j \preceq X^i$ if and only if $\{X_1^j, X_2^j, ..., X_q^j\} \subseteq \{X_1^i, X_2^i, ..., X_p^i\}$.

Then, a sub-problem $MOOP_i$ can be defined in the same manner as $MOOP$ in Equation (1) with a newly added

constraint set $\tilde{e}^i$:

$$\begin{array}{ll} \texttt{minimize} & \mathbf{y^i} = \mathbf{f^i}(\mathbf{x^i}) \\ \texttt{subject to} & \mathbf{e^i}(\mathbf{x}) = (\text{true}, ..., \text{true}) \\ & \wedge\ \tilde{\mathbf{e}}^{\mathbf{i}}(\mathbf{x}) = (\text{true}, ..., \text{true}) \\ \texttt{where} & \mathbf{x^i} \in \mathbf{X^i} \preceq \mathbf{X}, \mathbf{y^i} \in \mathbf{Y^i} \preceq \mathbf{Y}. \end{array} \quad (2)$$

Note that the (sub-)domain $\mathbf{X^i}$ of sub-problem $MOOP_i$ is derived from the domain $\mathbf{X}$ of $MOOP$ by dimension reduction. That is, only a fraction of the decision variables will be represented and considered. But every time one sub-problem is (sequentially) chained with other sub-problems, a new constraint set $\tilde{\mathbf{e}}^{\mathbf{i}}$ is transferred from the precedent sub-problem in the chain. That is, sub-solutions found in $MOOP_{i-1}$, i.e. $\mathbf{y^{i-1}}$, are formulated as a constraint set $\tilde{\mathbf{e}}^{\mathbf{i}}$ then passed to $MOOP_i$, prevent it from searching candidates that violate previous sub-solutions.

Our problem decomposition approach can be used in both *hierarchical* and *flat* manners. Hierarchical decomposition can be applied when the problem can be gradually refined, in a chain of sub-problems (also called top-down refinement), i.e., if $X_1 \preceq X_2 \preceq X_3 \preceq ... \preceq X_n$. On the other hand, sub-problems may not naturally expose such a relationship and completely be separated from each other. This would result in a *flat* decomposition, where sub-problems can simply be solved separately and simultaneously. Previous decomposition approaches [1], [12], [23] belong to this latter category.

Another metric characterizing the proposed decomposition is whether it harms the optimality of solution space or not. That is, if the solution space that contains one or more pareto solutions could be filtered out by the decomposition, it is called *lossy*. If there is no loss of the solution space during decomposition, in contrast, the method is called *lossless*. Lossless decompositions always guarantee the performance improvement while lossy ones would miss some pareto solutions.

B. Speculation

After decomposition, each sub-problem $MOOP_i$ can be solved individually by only considering $\mathbf{x^i}$. The evaluation function $\mathbf{f^i}$, however, cannot be computed only from $\mathbf{x^i}$ in some cases. Therefore, the fitness $\mathbf{f^i}$ has to be estimated with the partial information given in $\mathbf{x^i}$. Speculation is needed only for hierarchical decompositions that are typically lossy with respect to decision vectors, flat decompositions include completely independent sub-problems and speculations are not necessary.

C. Post-Optimization

When dealing with tightly coupled sub-problems, the solution space of individual sub-problems is reduced, i.e., what we call *lossy* decomposition. In this case, the global optimum of $MOOP$ may exist beyond the solution space of sub-solutions that individual sub-problems $MOOP_i$ cover. To prevent this, we propose to apply an additional optimization at the end, starting from the derived solution set. Any conventional MOEA can be used without any modification for this final optimization.

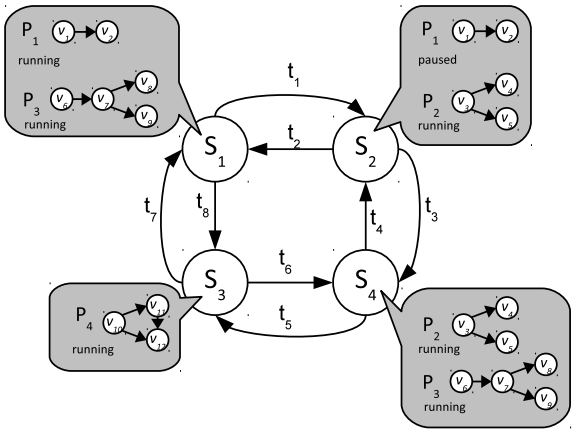

Fig. 3. Dynamic applications specified in KPNs and a controlling FSM.

IV. Proposed Mapping Optimization

With the problem decomposition presented above, the proposed mapping optimization of dynamic applications onto many-core systems [17] is illustrated in this section.

A. Problem description

Application: An application is described as a set of Kahn process networks (KPNs) [13], while the dynamic structure of the application is specified by a finite state machine (FSM). A KPN $\mathcal{P} = (V, L)$ consists of a set of processes V and a set of links L. An FSM $\mathcal{F} = (S, T, E, s_0, a, r, h)$ consists of a set of states S, a set of events E, a set of transitions $T \in S \times S$ and an initial state $s_0 \in S$. In addition, the three functions a, r, and h map transitions to sets of triggering events, states to sets of running process networks, and states to sets of paused (or halted) process networks, respectively. In other words, $a(t) \subseteq E$ for all $t \in T$, $r(s) \subseteq P$ for all $s \in S$ and $h(s) \subseteq P$ for all $s \in S$. Of course, $r(s) \cap h(s) = \emptyset$.

Fig. 3 presents an example $\mathcal{F}' = (S', T', E', s_1, a', r', h')$, with

- four states, among which s_1 is active initially, i.e., $S' = \{s_1, s_2, s_3, s_4\}$;
- eight transitions between states defined as follows: $T' = \{t_1, t_2, ..., t_7, t_8\}$ such that $t_1 = (s_1, s_2), t_2 = (s_2, s_1)$, and so on;
- the triggering events of each transition are defined in a', e.g., the transition t_2 from s_2 to s_1 happens when the event $a'(t_2) \in E'$ is detected in the state s_2;
- status of the process networks per each state, defined in r' and h' functions. For instance, in the state s_2, P_2 is running while P_1 is paused, i.e., $r'(s_2) = \{P_2\}$ and $h'(s_2) = \{P_1\}$.

Architecture: The target architecture is assumed to be hierarchical considering the recent design trend [14], [16], [18]. We maintain an abstract representation of this architecture, in a tree form, as depicted in Fig 4. For illustration, we restrict ourselves to only two levels of communication: (1) set of processors that form clusters communicating via a first level network, e.g., cluster (denoted cl_x in Fig 4(b)) and (2) clusters

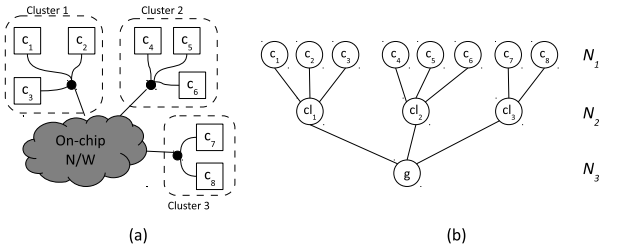

Fig. 4. (a) A two level hierarchical multiprocessor architecture and (b) its abstract representation.

connected via a second level network, e.g., on-chip network (g in Fig 4(b)).

To describe the architecture in Fig 4 in an abstract and formal manner, we propose the following formulation: The architecture $\mathcal{A} = (C, N^1, N^2, N^3)$ consists of a set of processors C and three sets of networks N^1, N^2, and N^3, which correspond to the three communication layers, i.e., intra-processor communication, intra-cluster communication, and inter-cluster communication. A network $n \in N^2$ is defined as the set of processors that are in the corresponding cluster, i.e. $n \subseteq C$. N^2 partitions the set of processors such that intra-cluster networks do not overlap, i.e. we have $\bigcup_{n \in N^2} = C$ and furthermore, $n \cap m = \emptyset$ for all $n, m \in N^2$, $n \neq m$. N^3 simply contains a single network $N^3 = \{n\}$ which contains all processors, i.e. $n = C$. N^1 contains a network for each processor as it represents the intra-processor communication, i.e. $|N^1| = |C|$ and for each $c \in C$ there is a network $n \in N^1$ with $n = \{c\}$. In particular, the communication architecture in Fig 4 is described by $N^2 = \{cl_1, cl_2, cl_3\}$ where $cl_1 = \{c_1, c_2, c_3\}$, $cl_2 = \{c_4, c_5, c_6\}$, and $cl_3 = \{c_7, c_8\}$. Without loss of generality, the level of communication can be extended although we assume two levels of communication in the following sections.

With the specification of *application* and *architecture* as above, we can define mapping problems as follows:

*1) **Decision Variables**:* Given the set of KPNs $K = \{\mathcal{P}_1 = (V_1, L_1), ..., \mathcal{P}_n = (V_n, L_n)\}$ and an FSM $\mathcal{F} = (S, T, E, s_0, a, r, h)$, for every pair of a process and a state that the process can be possibly in, determine the cluster cl and processor c binding. That is, for all (v, s) such that $(v \in V) \wedge (\mathcal{P} = (V, L) \in K) \wedge (s \in S) \wedge (\mathcal{P} \in r(s) \cup h(s))$,

$$\text{determine } (c, cl) \text{ such that } c \in cl \tag{3}$$

*2) **Constraints**:* A valid mapping should satisfy following constraints:

- No process migration: We need to make sure that a process is not mapped to different processors if the corresponding states are connected by a transition. For each constituent process $v \in V$ of a KPN $\mathcal{P} = (V, L)$ and each transition $t = (s_1, s_2) \in T$ of $\mathcal{F}$, the following condition should be kept for their decision variables (c_1, cl_1) and (c_2, cl_2) for (v, s_1) and (v, s_2):

$$\begin{aligned} &\mathcal{P} \in (r(s_1) \cup h(s_1)) \wedge \mathcal{P} \in (r(s_2) \cup h(s_2)) \\ &\longleftrightarrow cl_1 = cl_2 \wedge c_1 = c_2. \end{aligned} \tag{4}$$

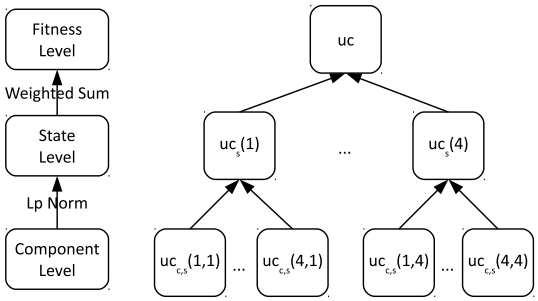

Fig. 5. Structure of fitness evaluation

- Processor utilization: Utilization of each processor is less than or equal to 1.0. The processor utilization $uc_{cs}(c, s)$ of a processor c for each state s should satisfy:

$$uc_{cs}(c, s) = \sum f(v) \cdot w(v, c) \leq 1.0, \tag{5}$$

where $f(v)$ and $w(v, c)$ are the number of firings per time unit and maximum execution time of process v on the processor c respectively and the mapping is determined to be (c, cl) for (v, s).

- Link bandwidth: The communication networks should be able to handle the aggregated bandwidth of all links mapped to them. The aggregated data volume for each link l in state s, $ul_{ls}(l, s)$, must be smaller than its supported rate $\sigma(l)$. Then, for $l \in L^2(\mathcal{P}, s, n) \cup L^3(\mathcal{P}, s, n)$,

$$ul_{ls}(l, s) = \sum \frac{f(l) \cdot q(l)}{\sigma(l)} \leq 1.0, \tag{6}$$

where $f(l)$ is number of firings (of the source node of l) per time unit, $q(l)$ is data volume per each firing, $\sigma(l)$ is maximum bandwidth of link l per time unit, and $L^{2/3}(\mathcal{P}, s, n)$ denotes the set of all links of $\mathcal{P} = (V, L)$ that are mapped to communication link $n \in N^{2/3}$ in state s.

*3) **Objective**:* The optimization goal of this problem is set to minimize the variance of workload between components. In other words, a mapping with well balanced workload is preferred over the unbalanced one for better throughput [21].

To quantify this, we use the L^p norm which is defined as follows for a vector $L = \{e_1, e_2, e_3, \ldots, e_n\}$:

$$L^p(L) = (\sum_{e \in L} e^p)^{1/p},$$

where a lower value indicates a better balanced one.

As we have three kinds of component (processor, cluster, and network) and they are stressed differently according to the current state of the FSM, we calculate the fitness value hierarchically as shown in Fig. 5. The norm values for all components in a state is calculated first and they are merged into a single representative value as a weighted sum. The weight vector w_s is set properly considering the average execution time of each state. For instance, L^p norm of processor utilization in state s is $uc_s(s) = L^p(\{uc_{cs}(c, s) | \forall c \in C\})$ and the weighted sum of all states is $uc = \sum_{s \in S} w_s \cdot uc_s(s)$. $ul^{(2)}$ and $ul^{(3)}$ can be calculated in a similar way.

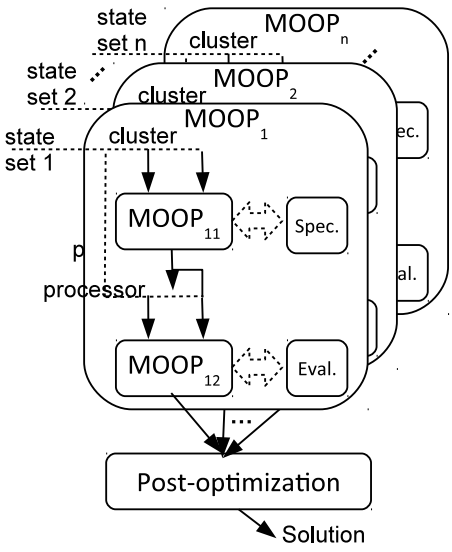

Fig. 6. Mapping optimization implemented with the decomposition framework.

In summary, the objective vector $\mathbf{f}$ to be minimized is

$$\mathbf{f} = (uc, ul^{(2)}, ul^{(3)}). \tag{7}$$

B. Decomposition

Fig. 6 depicts how the defined mapping optimization problem is structured in the proposed decomposition framework. Two decomposition approaches are applied in a mixed form. That is, the *state-based* one is applied as *flat* decomposition, each of which is decomposed further into sub-problems concerning the *architectural hierarchy* (*hierarchical* decomposition). Each will be discussed in detail in what follows.

1) State-based Decomposition: A process network may have more than one mapping, as it can be active in multiple scenarios with the constraint described in Eq. (4). That is, the mappings for the two connected states should be the same in order not to cause dynamic mapping adjustments. On the other hand, if a process network is active in two (or more) states and they are not directly reachable to each other, i.e., there is no single transition between states, it can have independent mappings optimized individually. In Fig. 3, $\mathcal{P}_1$ is active (or paused) in s_1 and s_2, where both of them are connected by a transition. We enforce the same mapping for two states here. On the other hand, $\mathcal{P}_3$ has two independently optimized mappings for s_1 and s_4.

To capture this relationship, we define *Connected State Groups* (CSGs) [17] for each application as the set of maximally connected states where the corresponding application is active (or paused). If two connected states have an active (or paused) application in common, they belong to the same CSG. $\mathcal{P}_1$ in Fig. 3, for instance, has only one CSG $\{s_1, s_2\}$, while $\mathcal{P}_3$ has two CSGs $\{s_1\}$ and $\{s_4\}$. CSGs may also have inter-relationships. CSG $\{s_1, s_2\}$ of $\mathcal{P}_1$ is related to CSG $\{s_2, s_4\}$ of $\mathcal{P}_2$ as they have $\mathcal{P}_2$ active in common in s_2. In other words, s_1 and s_2 are inter-related via $\mathcal{P}_1$ and so are s_2 and s_4 via $\mathcal{P}_2$. On the contrary, s_3 is isolated thus can be treated separately. Fig. 6 shows that independent sets, $S'_1, S'_2, ..., S'_n$, can be solved separately in $MOOP_1, MOOP_2, ..., MOOP_n$.

Thus, as the first decomposition, we separate states into several isolated sets, $\{s_1, s_2, s_4\}$ and $\{s_3\}$ in the example, and solve them independently. Then, for each sub-problem $MOOP_i$ with the state set $S' \subset S$, the decision variables in Eq. (3) is reduced to $s \in S'$ and the objective in Eq. (7) as well is partially evaluated only for $s \in S'$. There is no added constraint for this decomposition since the sub-problems are completely independent of each other. This is a *flat* and *lossless* decomposition.

2) Architecture-based Decomposition: Each $MOOP_i$ can further be decomposed considering the architectural hierarchy. We solve two decision variables, *task-to-cluster*(cl) and *task-to-processor*(c), separately and sequentially as shown in Fig. 6.

First, the *task-to-cluster* mapping is determined in $MOOP_{i1}$. The decision variable is now only cl in Eq. (3), while the objective is kept as Eq. (7). The sub-solution in this stage is now fed to the next sub-problem $MOOP_{i2}$ as a constraint. $MOOP_{i2}$ fixes *task-to-processor* while respecting the cluster mapping decision made above. That is, decision variables in $MOOP_{i2}$ is the same as Eq. (3) but with a newly added constraint $\tilde{e}^{i2}$ from the solution $MOOP_{i1}$, which is

$$c \in cl_{i1}. \tag{8}$$

The objective is kept as Eq. (7).

Since the *task-to-cluster* mapping phase does not have all the required information for the objective in Eq. (7), speculation is necessary as will be discussed later. The decomposed sub-problems in this case have a dimension reduction relationship ($X^{i1} \preceq X^{i2}$) and some solution space is lost due to the sequential chaining of them. Thus, this is a *hierarchical* and *lossy* decomposition.

The decompositions applied to the proposed mapping optimization are summarized in Table I.

TABLE I
DECOMPOSITIONS USED IN THE MAPPING OPTIMIZATION

	State-based	Architecture-based
Category	flat/lossless	hierarchical/lossy
Decomposition	independent states	architectural hierarchy
Speculation	not necessary	min-max bound
Post-optimization	not necessary	allowing inter-cluster mutation

C. Speculation

For a *task-to-cluster* mapping, $MOOP_{i1}$, in above decomposition, it is impossible to calculate the exact fitness value as the *task-to-processor* mapping is not yet fixed. Furthermore, a *task-to-cluster* decision implies various *task-to-processor* mappings. Another difficulty comes from the fact that the *monotonicity* property does not hold in this case in contrast to the previous decomposition approaches [1], [12]. To be more specific, a better choice at *task-to-cluster* mapping does not always cause a better fitness at *task-to-processor* mapping.

The fitness values of $\mathbf{f}$, uc and $ul^{(2)}$ in this case, should be *speculated* at this sub-problem. Without an actual mapping configuration, we can still calculate a bound that a fitness value

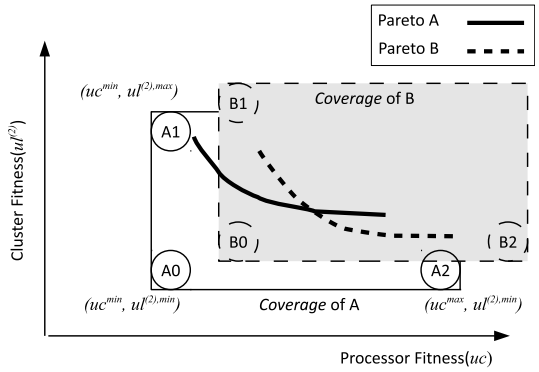

Fig. 7. The speculation in *task-to-cluster* mapping.

can possibly be in. For example, the minimum of uc is the case that all the workloads are as balanced as possible, which is denoted as uc^{min}.

Algorithm 1 illustrates how uc_s^{min} is speculated with the *task-to-cluster* mapping given. For brevity of the description, a cluster is assumed to have only homogeneous processors. However, this can easily be extended to the heterogeneous case without loss of generality. It basically follows the best-fit approach of the bin packing problem. That is, given the the *task-to-cluster* mapping, sort all the tasks that are mapped on a cluster cl (line 4-7). From the sorted list, the heaviest task is chosen (line 8) and assigned to the idlest processor in cl (line 13-14). If the heaviest one does not fit to the idlest processor, this solution should be marked as invalid. As the processor list c_{cl} should always be kept as sorted, the heap sorting is used here. $ul^{(2),min}$ can be calculated in a similar way. Likewise, the maximum values, uc^{max} can also be estimated considering the worst-case mapping. That is, the worst-fit approach of the bin packing problem is exploited here to calculate uc^{max} and $ul^{(2),max}$.

Algorithm 1 Speculation of uc_s^{min} at $MOOP_{i1}$

```
1:  for all cluster l do
2:    initialize a sorted array of processor utilization c_l[ ];
3:    initialize array v_l[ ];
4:    for all processes v_i do
5:      append v_i to v_l if cl_{v_i,s} = l;
6:    end for
7:    sort v_l by required utilization in descending order
8:    for j = 1 to |v_l| do
9:      if v_l[j] + c_l[1] > 1.0 then
10:       mark as invalid solution;
11:       return;
12:     else
13:       c_l[1] ← c_l[1] + v_l[j];
14:       sort c_l in ascending order;
15:     end if
16:   end for
17: end for
18: based on c_l[ ] obtained, calculate the L_p norm;
```

With the speculation of minimum and maximum values, we can define the *coverage* of a sub-solution as a space that the refined solutions from the sub-solution may fit in. The coverage of $MOOP_{i1}$, for instance, is a 2-dimensional rectangular space that is characterized by a combination of uc^{min}/uc^{max} and $ul^{(2),min}/ul^{(2),max}$ as shown in Fig. 7. Now that the problem is how to evaluate the coverages of two different sub-solutions.

One possible way is to use the minimum pair $(uc^{min}, ul^{(2),min})$, $A0$ and $B0$ in Fig. 7. We call this *min-min* speculation. But, this sometimes causes an inaccurate evaluation since the estimated point is too optimistic thus not achievable. The counter example is given in Fig. 7. Though $A0$ is better than $B0$ in Fig. 7 within *min-min* speculation, cluster-level mapping B provide better solutions at the end (as denoted as a dotted curve) in the case that smaller cluster fitness is more desirable than smaller processor fitness.

Alternatively, it can be allowed to have two different metrics: *min-max* $(uc^{min}, ul^{(2),max})$ and *max-min* $(uc^{max}, ul^{(2),min})$. One of those two is randomly selected at optimization and the choice does not change to the other unless the mapping decision is changed. The idea is to let both solutions have some possibility to survive if they are incomparable. If we choose A1 and B2 (or A2 and B1) in Fig. 7, for instance, two solutions are incomparable and then they both would survive. We call this *min-max* speculation.

D. Post-optimization

At the post-optimization procedure, we gather all the populations from each sub-solution and merge them into one. This merge process is also important in order to maximize the objective space coverage. Conventional solution selection methods such as SPEA-II [26] can be used at this stage. It is noteworthy that this selection procedure itself is already a huge problem. In addition, to compensate for the loss of solution space due to the decomposition, one more optimization can be conducted here. That is, intra- and inter-cluster mutations are enabled during the merging procedure.

V. Experiments

In this section, the proposed mapping optimization via the problem decomposition is evaluated quantitatively. The proposed technique is implemented in Java based on the PISA-EXPO [4] framework. All experiments have been conducted in a Linux machine that has eight processors (4 Dual-Core AMD Opteron 2218 processors) with 8GB main memory.

In what follows, we compare the proposed technique against a conventional MOEA (denoted as *EXPO*). Two decomposition techniques presented in Section IV-B1 and IV-B2 are applied to the base model in *SDEXPO* and *ADEXPO* respectively, while *SADEXPO* has both incorporated as shown in Fig. 6.

The parameters are set as follows: population size α, number of parent individuals μ, number of offsprings λ are all set to 100 (except *EXPO*) and three objectives in Eq. (7) are set to be minimized. SPEA-II [26] is used as a selector and constraint violations are avoided by imposing a big penalty on the fitness.

In the proposed decomposition approach, several MOEAs are separately running. Thus, to make fair comparisons, we let the base *EXPO* have more populations and iterations. That is, α, μ, and λ are all set to 500 and it is allowed to have 10 independent populations that will be merged at the end.

A. Limitation of Conventional Techniques

In advance to the main benchmarks, a synthetic KPN is tested here to show that the conventional MOEAs are not good enough to explore large scale solution space. The synthetic example consists of four very communication intensive KPNs within a single state. The target architecture is set to a two cluster system with each having eight cores inside. The objective is set to have balanced processor utilization and minimized communication on the network. One of the desired mapping solutions is that each KPN is isolated in a single cluster, triggering no communication overhead on the global network. This is a very intuitive mapping but hard to be found since it is one of the huge number of possible solutions. Furthermore, local minima make it harder to reach this optimum.

EXPO and *ADEXPO* are compared in the reachability. Both are tested 100 times and observed in how many times they have found out the solution. *EXPO* has only succeeded 3 times, while the success rate of *ADEXPO* exceeds 70%. This implies that the isolation of task-to-processor mapping enables more efficient exploration of the solution space. In particular, if the network utilization is of critical importance, the gain is more effective as intended in prioritization.

B. Benchmark Setup

We use two synthetic KPNs (*SynthSmall* and *SynthBig*) and a Picture-in-Picture (PiP) application for benchmarking. For the synthetic examples, the TGFF [6] is used to randomly generate the processes and links between them and the controlling FSM is designed arbitrarily. The PiP example, which is also shown in [17], consists of two MJPEG KPNs and an FSM.

The Intel Single-chip Cloud Computer (SCC) [14] that has 24 clusters on a single die is considered as target platform. In SCC, each cluster has two pentium cores integrated and they communicate with each other via a shared memory. A 4x6 2D-mesh on-chip network enables the inter-cluster communication. According to the benchmark size, we partially utilize the platform (from 8 to 48 cores). *SynthSmall* and *SynthBig* are tested on 8 and 12 cores, respectively, while we test two different architectures for *PiP*: one with 48 cores (*PiP_48*) and the other with 24 (*PiP_24*). To make the application fit into the 24 cores, the execution times of processes in PiP application are halved in *PiP_24*. More detailed information on the benchmarks is given in Table II.

C. Solution Space Reduction

First, let us investigate the characteristics of the solution space in above benchmarks. Even the smallest one, *SynthSmall* has total 1.33×10^{36} solutions to be explored. Table II shows that the solution space is dramatically reduced in the proposed decomposition technique. Note that not all of this solution space is valid mapping due to the constraints given in Eq. (4-6). To check how these constraint affect the exploration, we observe how often the basic *EXPO* fails to find a feasible solution within an hour. In *SynthSmall* and *PiP_48*, 10 and 4 out of 20 trials have been failed while others have always been successful. Therefore, we can state that *SynthSmall* and *PiP_48* are more constrained than others. In the proposed technique, on the other hand, we have always succeeded to find feasible solutions within an hour in all benchmarks.

D. Improvement on Optimality

To quantitatively measure the effectiveness of the proposed technique, the output pareto curves are mainly evaluated in two metrics: hypervolume and Inverted Generational Distance (IGD) [27]. The hypervolume of a pareto curve is defined as area of the dominated space by the curve in the domain, while IGD denotes the average euclidean distance of solutions to the true pareto. The bigger hypervolume we have, the more optimized a solution is, while smaller IGD indicates more varied solution.

Hypervolumes of the optimized curves (normalized to *EXPO*) are shown in Fig. 8(a). As the state-based decomposition is lossless, *SDEXPO* has never shown worse optimality than *EXPO*. In *SynthSmall*, all the states are inter-related thus no state-based decomposition is applicable. Note that the solution space of *SDEXPO* remains the same as *EXPO* for *SynthSmall* in Table II.

The architecture-based decomposition technique is verified in *ADEXPO*. It is a lossy decomposition, which is probable to lose some good solution space during the decomposition. Inaccurate speculation may worsen this. The *min-min* speculation in *ADEXPO* for *SynthSmall* illustrates this side effect, where the hypervolumes are even less than the base *EXPO*. Alternatively, the *min-max* speculation can be used to overcome this shortcoming. In all benchmarks, it enables larger design space exploration as indicated in larger hypervolumes. This emphasizes the necessity of accurate speculations.

To take a deeper look at the effect of the architecture-based decomposition, we project the obtained pareto curves of *SynthBig* from *EXPO* and *SADEXPO* onto the 2-dimensional space of uc and $ul^{(3)}$ as shown in Fig. 9. It is clearly shown that *SADEXPO* could explore more solution space below a certain fitness level (0.27) in the network fitness $ul^{(3)}$. This result complies with the intended priortization which puts $ul^{(3)}$ first in the sub-problem chain.

Having both decomposition techniques incorporated together, *SADEXPO* always shows the biggest hypervolume. *SADEXPO* can effectively increase the hypervolume from 6.9% in *PiP_48* to 27.5% in *SynthBig* compared to *EXPO*. The normalized IGDs are compared in Fig. 8(b) to show the diversity of solutions. The values are smaller with the decomposition techniques except for *SynthSmall* with *min-min* speculation. Again, this is due to the side effect of the inaccurate speculation.

TABLE II
DESCRIPTION OF BENCHMARKS

	#State	#Application	#Processes	#Clusters	#Processors	Problem Space	Problem Space Reduction		
							SDEXPO	ADEXPO	SADEXPO
SynthSmall	2	2	40	4	8	1.33E+36	1.33E+36	1.21E+24	1.21E+24
SynthBig	5	5	100	3	12	1.22E+151	8.28E+107	1.94E+84	1.61E+60
PiP_24	9	3	159	12	24	6.58E+249	2.77E+150	2.88E+163	2.73E+98
PiP_48	9	3	159	24	48	2.02E+304	1.80E+183	6.58E+249	2.77E+150

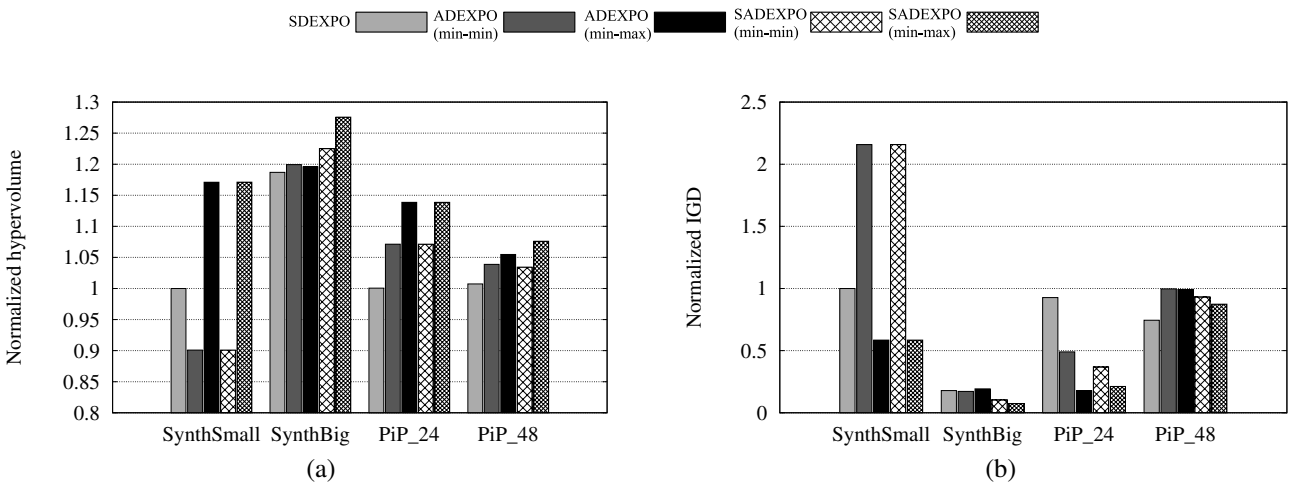

Fig. 8. Comparative evaluation of optimized pareto curves (normalized to *EXPO*): in (a) hypervolume and (b) IGD.

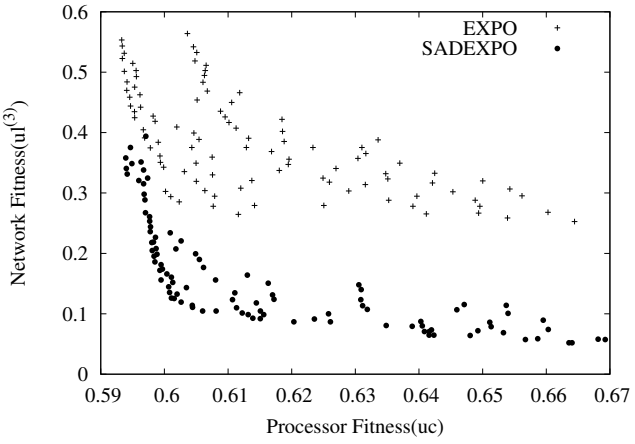

Fig. 9. Projected pareto curve of *SynthBig* benchmark in *EXPO* and *SADEXPO*.

We have compared the pareto curves from *EXPO* and *SADEXPO* further by counting the number of points that are non-dominated by each other as shown in Table III. In *SynthBig* and *PiP_24*, *SADEXPO* mostly dominates the curve from *EXPO*. On the other hand, in highly constrained benchmarks, *SynthSmall* and *PiP_48*, considerably many solutions from *EXPO* survive as non-dominated against *SADEXPO*. This is explained in Fig. 10, where the pareto solutions obtained from *EXPO* for *PiP_48* are depicted. The non-dominated solutions are highlighted properly. It is shown that most non-dominated points tend to have lower cluster fitness value. The dominated ones have lower network fitness but still higher than *SADEXPO*. Therefore, it can be analyzed that most gain for hypervolume is from the network fitness and the decomposition negatively influences the cluster fitness. This exemplifies the cost of the proposed technique. That is, the abstraction and prioritization enable larger design space exploration (higher hypervolume and lower IGD) at the cost of probable sacrifice in some objectives (cluster fitness in this case).

TABLE III
THE PORTION OF NON-DOMINATED SOLUTIONS AGAINST EACH OTHER

	EXPO	SADEXPO
SynthSmall	0.639	0.784
SynthBig	0.05	1.000
PiP_24	0.156	0.980
PiP_48	0.96	0.42

E. Effect of Post-Optimization

Lastly, the effect of the post-optimization is evaluated separately. We have compared the solutions that are not yet fed to the post-optimization step with the final solutions and counted the number of dominated points. The maximum gain is achieved in *PiP_24*, where 92% of the solutions are dominated by the final curve. This means that 92% of the sub-solutions have been improved further by the post-optimization. Even in the worst-case, *PiP_48*, 13% of the populations have been improved by the post-optimization.

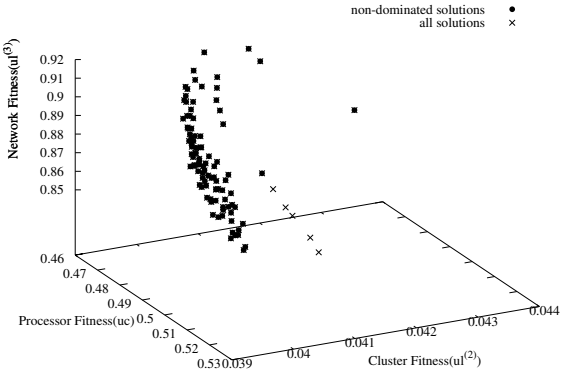

Fig. 10. Pareto front of *PiP_48* obtained from *EXPO*. Non-dominated solutions by *SADEXPO* are highlighted.

VI. Conclusion

In this paper, we propose an efficient multi-objective mapping optimization technique for many-core systems. To deal with the vast solution space, the problem is divided into several smaller sub-problems and solved individually. To break the inter-relationship between them, speculations are conducted for fitness evaluation. Separately obtained sub-solutions are merged into a complete one at the end, where one more global optimization is performed. Experimental results show that the decomposed technique outperforms the traditional meta-heuristics both in optimality and diversity. The proposed problem decomposition technique can also be applied to other large scale EDA optimization problems.

Acknowledgment

This work was supported by EU FP7 project EURETILE (grant number 247846), Korean-Swiss science and technology cooperation program, and National Research Foundation of Korea (NRF-2011-357-D00213).

References

[1] S. G. Abraham, B. R. Rau, and R. Schreiber. Fast design space exploration through validity and quality filtering of subsystem designs. Technical report, Packard, Compiler and Architecture Research, HP Laboratories Palo Alto, 2000.

[2] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based noc architectures. In *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on*, pages 182 – 187, sept. 2004.

[3] S. Bekiroğlu, T. Dede, and Y. Ayvaz. Implementation of different encoding types on structural optimization based on adaptive genetic algorithm. *Finite Elements in Analysis and Design*, 45(11):826–835, 2009.

[4] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. Pisa—a platform and programming language independent interface for search algorithms. In *Evolutionary multi-criterion optimization*, pages 1–1. Springer, 2003.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2), 2002.

[6] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *CODES/CASHE '98: Proceedings of the 6th international workshop on Hardware/software codesign*. IEEE Computer Society, Mar. 1998.

[7] N. Eklund, M. Embrechts, and M. Goetschalckx. Efficient chromosome encoding and problem-specific mutation methods for the flexible bay facility layout problem. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(4):495–502, 2006.

[8] F. Ferrandi, C. Pilato, D. Sciuto, and A. Tumeo. Mapping and scheduling of parallel c applications with ant colony optimization onto heterogeneous reconfigurable mpsocs. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 799 –804, jan. 2010.

[9] C. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele. Solving hierarchical optimization problems using moeas. 2003.

[10] M. Geilen and T. Basten. Reactive process networks. In *Proceedings of the 4th ACM international conference on Embedded software*, EMSOFT '04, pages 137–146, New York, NY, USA, 2004. ACM.

[11] E. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 5(2):113 –120, feb 1994.

[12] J. R. Josephson, B. Chandrasekaran, M. Carroll, N. Iyer, B. Wasacz, G. Rizzoni, Q. Li, and D. A. Erb. An architecture for exploring large design spaces. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 143–150, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[13] G. Kahn. The semantics of a simple language for parallel programming. *proceedings of IFIP Congress74*, 1974.

[14] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core scc processor: the programmer's view. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[15] R. Moritz, T. Ulrich, L. Thiele, and S. Brklen. Mutation operator characterization: Exhaustiveness, locality, and bias. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1396–1403, New Orleans, USA, 2011.

[16] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[17] L. Schor, I. Bacivarov, D. Rai, H. Yang, S. haeng Kang, and L. Thiele. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *CASES*, 2012. To appear.

[18] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):18:1–18:15, Aug. 2008.

[19] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping applications to tiled multiprocessor embedded systems. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*, pages 29 –40, july 2007.

[20] H. Yang and S. Ha. Ilp based data parallel multi-task mapping/scheduling technique for mpsoc. In *SoC Design Conference, 2008. ISOCC '08. International*, volume 01, pages I–134 –I–137, nov. 2008.

[21] H. Yang and S. Ha. Pipelined data parallel task mapping/scheduling technique for mpsoc. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 69 –74, april 2009.

[22] Y. Yi, W. Han, X. Zhao, A. Erdogan, and T. Arslan. An ilp formulation for task mapping and scheduling on multi-core architectures. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 33 –38, april 2009.

[23] Q. Zhang and H. Li. A multi-objective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation, Accepted*, 2007, 2007.

[24] W. Zhou, Y. Zhang, and Z. Mao. Pareto based multi-objective mapping ip cores onto noc architectures. In *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, pages 331 –334, dec. 2006.

[25] E. Zitzler, D. Brockhoff, and e. al. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. *Evolutionary Multi-Criterion Optimization*, 2007.

[26] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.

[27] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.