Integrated Task Clustering, Mapping and Scheduling for Heterogeneous Computing Systems

Yuet Ming Lam
Macau University of Science and Technology
ymlam@must.edu.mo

Abstract

This paper presents a new approach for mapping and scheduling task graphs for heterogeneous hardware/software computing systems using heuristic search. Task mapping and scheduling are vital in hardware/software codesign and previous approaches that treat them separately lead to suboptimal solutions. In this paper, we propose two techniques to enhance the speedup of mapping/scheduling solutions: (1) an integrated technique combining task clustering, mapping, and scheduling, and (2) a multiple neighborhood function strategy. Our approach is demonstrated by case studies involving 40 randomly generated task graphs, as well as six applications. Experimental results show that our proposed approach outperforms a separate approach in terms of speedup by up to 18.3% for a system with a microprocessor, a floating-point digital signal processor, and an FPGA.

Keywords Hardware/software codesign; heuristic search; multiple neighborhood functions

1 INTRODUCTION

Digital signal processing (DSP) algorithms are often computationally intensive and contain different degrees of parallelism as well as different mixes of arithmetic operations. Such systems require increasingly greater amounts of processing power and place new demands on the computation hardware. Heterogeneous computing systems containing software processors (e.g. microprocessors), hardware processors (e.g. reconfigurable hardware), and dedicated DSP processors provide potentially more effective solutions than single microprocessor systems for many real time and embedded DSP applications. Reconfigurable hardware, such as field programmable gate arrays (FPGAs), contains reconfigurable fabric upon which custom functional units can be built. Time-critical tasks can be executed on the reconfigurable hardware, taking advantage of spatial parallelism to achieve high performance. Microprocessors, which often operate with a higher clock rate than reconfigurable hardware, execute sequential tasks more efficiently. Dedicated DSP processors, on the other hand, provide an efficient way to execute specific arithmetic operations.

Hardware/software codesign takes an application specification as input, and generates an implementation for a heterogeneous computing system that fulfils given performance criteria, e.g. minimise the execution time of the application on the hardware platform. Codesign majorly involves partitioning, mapping, and scheduling. Partitioning divides the entire application into a set of smaller tasks, and a task graph is often constructed to represent the data/control precedences among tasks. Mapping process thus assigns tasks to processing elements and scheduling process determines the execution sequence of tasks. This work focuses on mapping and scheduling. Finding the best mapping and scheduling given a task graph is known to be NP-hard in its general form and previous work proposed to address this issue is shown in Table 1.

A list scheduling [32] approach that builds partial valid solutions until a complete solution is formed has been widely used. In list scheduling, each task is assigned a priority based on heuristics. Available tasks are scheduled in each iteration based on the assigned priority, such as critical path [1],[2], job length [4], and number of successors [5]. A review of such list scheduling techniques for homogeneous systems is given in [3]. However, the calculated priorities may lose their meaning for a heterogeneous system, so such techniques are modified to address heterogeneous systems, e.g. using the maximum [6], median [7], or mean processing time [8]. Approaches that considering multiple criteria are also proposed, examples are using earliest starting and finishing time [9], latest finishing time and earliest starting time [10]. Another approach involves a deterministic search-based method. This approach generates an initial solution using list scheduling, and then applies a heuristic to move tasks between processors to find a better solution. For instance, moving critical tasks to faster processors [11], Kim et. al. propose a Push-Pull technique to reduce the idle periods of processors [13]. To shorten the search time, a topological order based task selection is proposed [12].

TABLE 1
Some approaches to address mapping/scheduling.

approach	reference	examples of applications	comments
list scheduling	[1][2][3]	Random graphs, tracking algorithms	For homogeneous systems
	[4]	Navigation system	Shortest job first
	[5]	FFT	Scheduling based on depth of task and number of successors
	[6][7][8]	Random graphs, FFT	Use maximum/mean/median execution time to calculate priority
	[9][10]	Random graphs, FFT	Use starting/finishing time to calculate priority
deterministic search	[11]	Random graphs	Iteratively move critical task from software processor to hardware processor
	[12]	Random graphs, Gaussian elimination	Selecting tasks based on topological order, move task between processors, for homogeneous systems
	[13]	Random graphs	Move tasks to fill the idle periods of processors
heuristic search	[14][15]	Random graphs	Genetic algorithm, simulated annealing, address mapping only
	[16][17][18]	Random graphs, FFT, JPEG	Separate mapping and scheduling
	[19][20][21] [22][23]	Random graphs, mean value analysis	Combine mapping and scheduling
	this work	Random graphs, 5 real applications	Integrate clustering, mapping, and scheduling, compare two heuristic search methods
integer linear programming	[24]	Filtering	For VLIW architecture
	[25][26]	Random graphs, FFT, smith waterman	For reconfigurable hardware. Limited number of tasks
	[27]	Random graphs	15 tasks maximally, for homogeneous systems

	[28]	Packet forwarding	Relax the integer constraint
	[29]	AES, DES	For reconfigurable hardware, choose among ILP and list-scheduling
search method comparison	[30]	Synthetic	Address mapping only, assume zero communication cost.
	[31]	Random graphs	Separate mapping and scheduling

Heuristic search techniques [33] are also applied to tackle the mapping and scheduling problem. In [14], the mapping process uses a genetic algorithm without considering scheduling. A similar approach is also proposed using simulated annealing [15]. Heuristic search has been applied to find the best mapping while a list scheduling method estimates the total execution time [16],[17],[18]. However, these approaches separate mapping with scheduling and this may result in sub-optimal solutions. Alternative approaches are thus proposed that combine mapping and scheduling [19],[20],[21],[22],[23]. In these approaches, a mapping/scheduling solution is modeled as assigning each processor an ordered job list to be executed. Mapping and scheduling are thus combined as a single process that assigns tasks to these job lists, i.e. mapping and scheduling of a task is determined after assigning this task to a job list. The impacts of various heuristic search methods on mapping/scheduling quality and search time has also been analyzed [30],[31]. However, only mapping is addressed and zero communication cost is assumed in [30] which is not realistic. In [31], mapping and scheduling are considered independently, and heuristic search techniques are only applied to find the best mapping.

Integer linear programming (ILP) is also applied to solve the mapping/scheduling problem [24]. This method guarantees an optimal solution but lacks scalability, the problem size of applications analyzed is often limited [25],[26]. The applicability of such an approach is studied in [27], but the maximum number of task analyzed is only 15 because of the long search time. To reduce search time, a technique that relaxes the integer constraint is proposed [28]. However, an additional process of rounding the fractional numbers in the found solution must be carried out to construct a feasible solution. This process may generate a solution which is having a totally different characteristic. A hybrid approach that consists of an ILP and a list based scheduler is proposed in [29]. While the ILP scheduler is chosen for simpler problems, the list scheduler is used for complex problems.

Compared with an approach that considers mapping and scheduling independently, combining mapping and scheduling covers a larger search space that results in a higher chance of covering good solutions. However, the extended search space can also increase the difficulty of finding good solutions as the search space grows exponentially with the problem size. Previous approaches use either a separate approach or combining mapping and scheduling [19],[20]. In this work, a new integrated approach that combines clustering, mapping, and scheduling is presented. The main contributions are as follows:

- A strategy with multiple neighborhood functions to enhance the speedup of mapping/scheduling solutions (Section 2.2).
- A task clustering technique to reduce data transfer overhead (Section 3.4).
- Integrating task clustering, mapping, and scheduling in a single process (Sections 3.2 and 3.4).
- An analysis of different neighborhood functions and two heuristic search techniques, simulated annealing and tabu search (Section 4.4).

The remainder of this paper is organised as follows: Section 2 provides an overview of the mapping/scheduling system and the multiple neighborhood functions based strategy. The design of neighborhood functions and the integrated clustering, mapping, and scheduling technique are introduced in Section 3. Experimental results are presented in Section 4, and finally, concluding remarks are given in Section 5.

2 MAPPING AND SCHEDULING

2.1 Overview

Figure 1 shows an overview of the approach used to find best mapping/scheduling solutions for applications described as directed acyclic task graphs. Given a task graph and a target architecture specification which includes processing elements and communication channel characteristics, a heuristic search is used to generate the best solution. In particular, different mapping/scheduling solutions (neighbors) are generated using multiple neighborhood functions iteratively, and an overall processing time, which is the time to finish all tasks, is calculated for each solution and used as the cost to guide the search. The goal is to find a solution with minimum overall processing time.

2.2 Multiple neighborhood functions

Heuristic search techniques can be applied to solve combinatorial optimization problems even though they do not guarantee optimal solutions. The basis of these techniques is neighborhood search, which starts with a feasible solution and attempts to improve it by searching its neighbors, i.e. solutions that can be reached directly from the current solution by an operation called a move [34]. This process is repeated until a local optimum or the termination condition is reached. Examples of advanced neighborhood search techniques are simulated annealing and tabu search [35]. In these techniques, the design of a neighborhood function, also known as a move function in the literature, is vital and there is no standard solution [33].

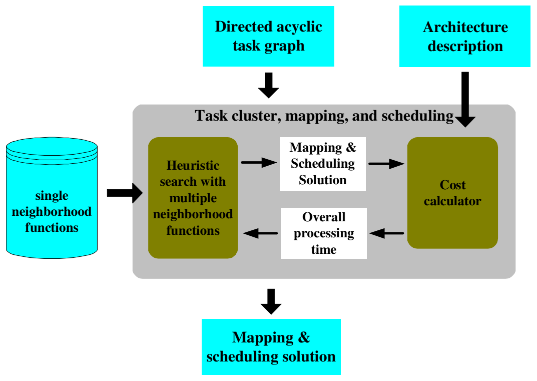
Directed acyclic task graph
Architecture description
Task cluster, mapping, and scheduling
single neighborhood functions
Heuristic search with multiple neighborhood functions
Mapping & Scheduling Solution
Overall processing time
Cost calculator
Mapping & scheduling solution

Fig. 1. Overview of the clustering, mapping, and scheduling system with multiple neighborhood functions.

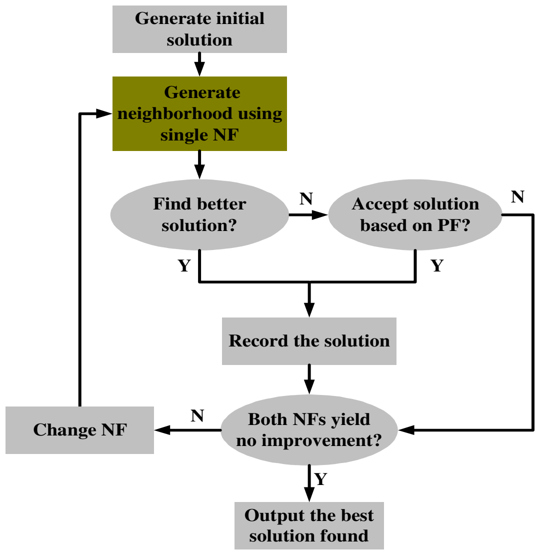
Generate initial solution
Generate neighborhood using single NF
Find better solution?
N
Accept solution based on PF?
N
Y
Y
Record the solution
Change NF
N
Both NFs yield no improvement?
Y
Output the best solution found

Fig. 2. Simulated annealing with multiple neighborhood function strategy. PF: probability function, NF: neighborhood function.

Simulated annealing accepts not only neighbors that improve on the previously best solutions, but also those that increase the cost. The following probability function is used to determine whether to accept a neighbor:

$$p = exp((cost(b) - cost(c)) / T) \qquad (1)$$

where b is the best neighbor and c is the current neighbor, T is called the temperature and is updated as: T =αT, where α is a constant and typically in the range of 0.9 to 0.99. Instead of using a single neighborhood function, a strategy involving multiple neighborhood functions is proposed. As shown in Figure 2, a feasible solution is initially generated and a neighborhood function is chosen to generate a new neighbor. If the new neighbor yields lower cost than the best previous solutions, it is accepted, i.e. a move from previous solution to the new one. Otherwise, acceptance of this new neighbor depends on the probability function. If there is no improvement after iterating a given number of times, a new neighborhood function is chosen to replace the old one. In this work, two neighborhood functions are used alternately, and the search ends when neither neighborhood function produces better solutions.

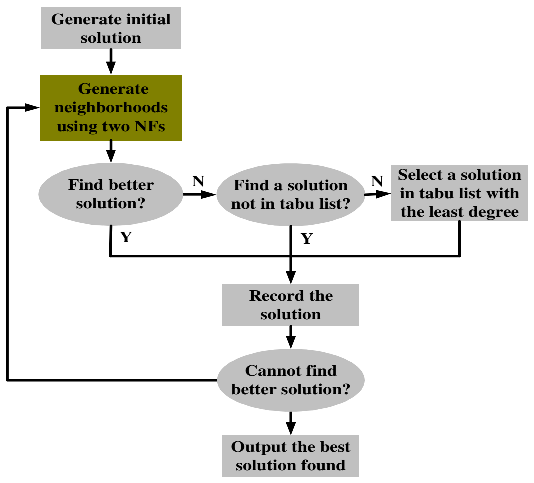
Generate initial solution
Generate neighborhoods using two NFs
Find better solution?
N
Find a solution not in tabu list?
N
Select a solution in tabu list with the least degree
Y
Y
Record the solution
Cannot find better solution?
Output the best solution found

Fig. 3. Tabu search with multiple neighborhood function strategy. NF: neighborhood function.

Tabu search keeps a list of the searched space and uses it to guide the future search direction; it can forbid the search moving to some neighbors. The proposed tabu search is based on multiple neighborhood functions (Figure 3). After an initial solution is generated, two neighborhood functions are used to generate neighbors simultaneously. If there exists a neighbor of lower cost than the best solution so far and it cannot be found in the tabu list, this neighbor is recorded. Otherwise a neighbor that cannot be found in the tabu list is recorded. If all the above conditions cannot be fulfilled, the solution in the tabu list with the least degree, i.e. the solution being resident in the tabu list for the longest time, is recorded. If the recorded solution has a smaller cost than the best solution so far, it is recorded as the best solution. The searched neighbors are added to tabu list and solutions with the least degree are removed. This process is repeated until the search cannot find a better solution for a given number of iterations.

3 METHODOLOGY

3.1 Directed acyclic task graph

Given a set of tasks TK = {tk_1, tk_2, ..., tk_n} to be executed, a directed acyclic graph can be defined as G = (TK, DF). Each task tk_i is a node in the graph and DF = {df_{ij} ; i = 1, 2, ..., n; j = 1, 2, ..., n} is a set of directed edges corresponding to data flow dependencies between tasks. Each edge df_{ij} 2 DF denotes an amount of data flow from task tk_i to tk_j , so tk_i is a predecessor of tk_j , and conversely, tk_j is a successor of tk_i. The number of nodes and edges are defined as |TK| and |DF| respectively. Given a set of interconnected heterogeneous processing elements PE = {pe_1, pe_2, ..., pe_m}, each task node tki of the task graph is associated with a set {t_{i1}, t_{i2}, ..., t_{im}} that denotes the execution times for implementing this task on each processing element of PE. The time to transfer results between tasks can be represented as DT = {dt_{ij} ; i = 1, 2, ..., n; j = 1, 2, ..., n}, each dt_{ij} denoting the time to transfer results from task tk_i to tk_j . This is calculated as the amount of data flow df_{ij} divided by the data transfer rate.

3.2 Integrated mapping and scheduling

In the integrated approach, mapping and scheduling are performed as a single step. Given a set of task lists PL = {pl_1, pl_2, ..., pl_m}, where pl_j = (as_{j1}, as_{j2}, ..., as_{jq}) is an ordered task sequence to be executed by processing element pe_j, each task in pl_j will be processed by pe_j in sequence when it is ready for execution, in other words, when all of its predecessors are finished. Task mapping and scheduling thus deal with assigning tasks to task lists. A task assignment function is defined as A: TK → PL, e.g. A(tk_i) = as_{jk} denotes task tk_i being assigned to as_{jk} of list pl_j. This means that tk_i is the *kth* task to be executed by processing element pe_j. A

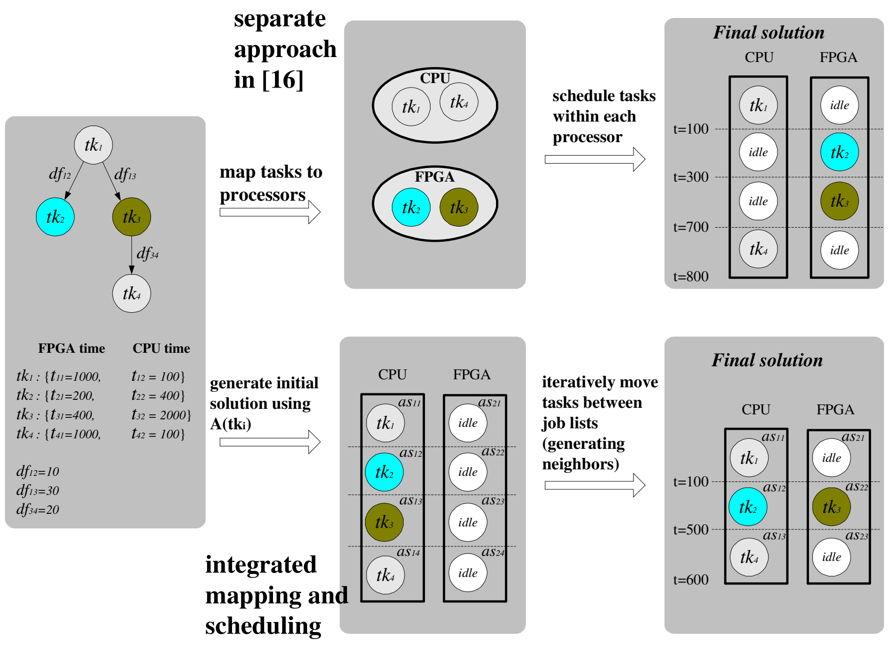

Fig. 4. An example of a directed acyclic task graph and its mapping/scheduling solutions using the separate approach in [16] and an integrated mapping and scheduling approach (Section 3.2), where t_{i1} and t_{i2} denote the execution time of task tk_i on FPGA and CPU respectively, and df_{ij} is the amount of data flow between task tk_i and tk_j. In the separate approach, due to inappropriate task mapping, where tk_1 and tk_4 are mapped to the CPU and the others to the FPGA, a poor scheduling solution is generated that execute tasks tk_2 and tk_3 in sequential order. In the integrated mapping and scheduling approach, as the mapping and scheduling of tasks are less constrained, a better solution is generated after moving task tk_3 from the CPU to FPGA.

mapping/scheduling solution is characterized by assignments of all tasks to processing elements, i.e. for every task tk_i in TK, A(tk_i) = as_{jk} for a pl_j in PL. It is assumed that only one task can be assigned to each as_{ij} . Based on this formulation, simulated annealing and tabu search based on multiple neighborhood functions are then applied to modify the task assignment and generate mapping/scheduling solutions (neighbors) iteratively.

In the separate approach, tasks are first mapped to processing elements, and each processing element contains a set of un-ordered tasks to be executed. A list scheduling technique is then used to determine the execution orders of tasks. For instance, the best task mapping can be found using heuristic search techniques, and a longest-execution-time-first scheduling method is used to determine the execution orders of tasks [16]. In contrast to the separate approach with constrained execution order, execution orders of tasks can be changed in the integrated approach. The search space of the integrated approach is much larger and results in a higher chance of covering good solutions. Figure 4 illustrates an example in which a task graph and the corresponding mapping/scheduling solutions using the separate approach in [16] and the integrated approach, where t_{i1} and t_{i2} are the execution times of task tk_i on an FPGA and CPU respectively. In the separate approach, due to inappropriate task mapping, where tk_1 and tk_4 are mapped to the CPU and the others to the FPGA, a poor scheduling solution is generated that executes tasks tk_2 and tk_3 in sequential order. In the integrated mapping and scheduling approach, as the mapping and scheduling of tasks are less constrained, a better solution is generated after moving task tk_3 from the CPU to FPGA.

Since the search space of the integrated approach is much larger, finding good solutions can be more challenging. A multiple neighborhood function strategy is thus proposed to increase the diversification of solutions and help to find better solution. Details of this strategy are introduced in Section 2.2.

RanRlo: random task relocation
step (1) Randomly select a task from list pl_p position as_{pq}.
step (2) Randomly select another task at list pl_j position as_{jk}.
step (3) Relocate task from (pl_p, as_{pq}) to (pl_j , as_{jk}).

GudRlo: guided task relocation
step (1) Randomly select a task at list pl_p position as_{pq}.
step (2) For all lists pl_j in PL and positions as_{jk}:
    Relocate task (pl_p, as_{pq}) to (pl_j, as_{jk}) which yields lowest cost.

RanRloClu: random relocation with task clustering
step (1) Perform step (1) to (3) of RanRlo.
step (2) Relocate largest data flow parent of task (pl_p, as_{pq}) to a position in pl_j which yields lowest cost.
step (3) Accept such relocation if the cost is lower.

GudRloClu: guided relocation with task clustering
step (1) Perform step (1) to (2) of GudRlo.
step (2) Relocate largest data flow parent of task (pl_p, as_{pq}) to a position in pl_j which yields lowest cost.
step (3) Accept such relocation if the cost is lower.

RanSwp: random task swap
step (1) Randomly select a task at list pl_p position as_{pq}.
step (2) Randomly select another task at list pl_j position as_{jk}.
step (3) Swap task in (pl_p, as_{pq}) with task in (pl_j , as_{jk}).

GudSwp: guided task swap
step (1) Randomly select a task at list pl_p position as_{pq}.
step (2) For all lists pl_j in PL and positions as_{jk}:
    Swap task (pl_p, as_{pq}) with task (pl_j, as_{jk}) which yields lowest cost.

BstRlo: best task relocation
step (1) For all lists pl_p in PL and positions as_{pq}:
    Relocate task (pl_p, as_{pq}) to every list pl_j in PL and position as_{jk},
setp (2) Choose the relocation which yields lowest cost.

Fig. 5. Seven basic neighborhood functions used in this work. Neighborhood functions RanRloClu and GudRloClu further integrate clustering process with mapping and scheduling by employing step (2) in these two functions.

3.3 Neighborhood functions

Given that s denotes the current mapping/scheduling solution, Figure 5 shows the seven basic neighborhood functions, NFx(s), adopted. RanRlo and GudRlo accept a mapping/scheduling solution; relocate a task from one task list to another and form a new neighbor. While RanRlo relocates tasks randomly, GudRlo chooses the best relocation position for a randomly selected task. In contrast to these two relocation based neighborhood functions, RanSwp and GudSwp swap the positions of two tasks and generate a new neighbor. In particular, RanSwp swaps the positions of two randomly selected tasks, and GudSwp randomly selects a task and chooses the best swapping position for this task.

Apart from the neighborhood functions introduced above, one more neighborhood function, BstRlo, which uses a best-relocation strategy [19], is analysed. This neighborhood function searches all possible relocations for all tasks, and chooses the one which yields lowest cost as the new neighbor.

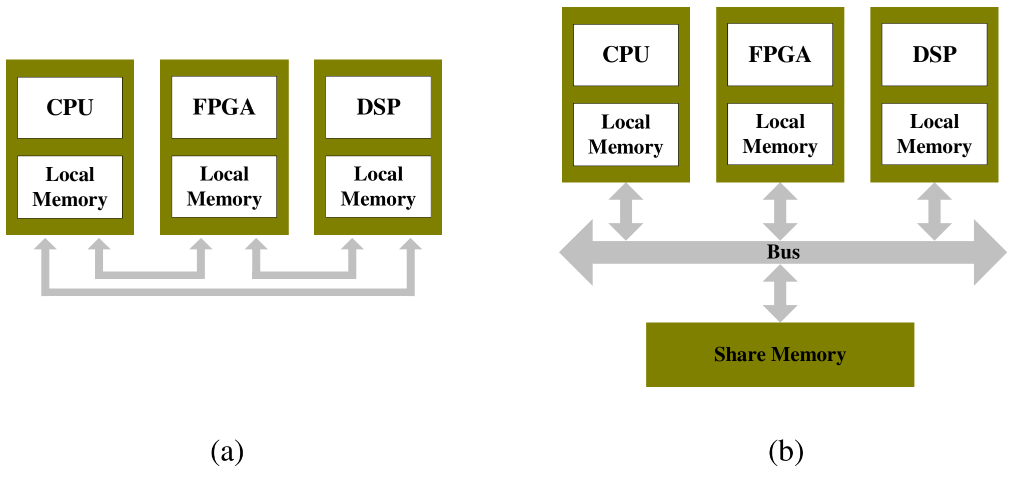

Fig. 6. Two reference architecture models of heterogeneous computing systems: (a) Loosely-coupled architecture with fully-connected communication. (b) Tightly-coupled architecture with bus

3.4 Integrated clustering, mapping, and scheduling

Based on the observation that mapping tasks with large data flow to the same processing element can potentially reduce data transfer overhead, a clustering technique, which attempts to group tasks with large data flow and allocate them to the same processing elements has been proposed [16]. However, clustering, mapping, and scheduling were solved separately in previous works, which give sub-optimal results. This work thus proposes two new neighborhood functions, RanRloClu and GudRloClu, which integrate the clustering technique into the neighborhood function, i.e. after relocating a task to another task list, move the parent which has the largest data flow among all parents of the current task to the same task list. If such a move yields lower cost, we accept this move and generate a new neighbor, otherwise, discard it. As mapping and scheduling have already been integrated (Section 3.2), by further combining clustering, neighborhood functions RanRloClu and GudRloClu thus integrate clustering, mapping, and scheduling in a single neighborhood function.

3.5 Cost function

As mentioned before, overall processing time is used as the cost to guide the heuristic search. This is the time for processing all tasks using the reference heterogeneous computing system and includes data transfer time. The processing time of a task tk_i on processing element pe_k is calculated as the execution time of tk_i on pe_k plus the time to retrieve results from all of its predecessors. The data transfer time between a task and a predecessor is assumed to be zero if they are assigned in the same processing element.

It is noted that tasks cannot be arbitrarily moved due to data dependencies, i.e. a task cannot be moved to a location such that it will execute prior to its predecessor. To avoid generating such infeasible solutions, our approach inflicts a large penalty cost in such cases.

4 RESULTS

4.1 Reference architecture

Multiprocessors systems can be tightly-coupled or loosely-coupled [36]. In a tightly-coupled architecture, shared memory is used to communicate between processing elements, data sent from one processing element are buffered in shared memory before being read by another processing element. However, shared memory is not used in a loosely-coupled architecture. Data are sent from one processing element to another directly without buffering. In this work, performance of the proposed mapping/scheduling system is evaluated on both architectures although generalisations are possible.

Figure 6 shows the two reference heterogeneous computing systems analysed: a loosely-coupled architecture with fully-connected communication having low communication contention, and a tightly-coupled architecture with bus communication having high communication contention. Both systems contain three processing elements: one microprocessor, one FPGA, and one DSP processor. Each processing element has a local memory for data storage during task execution, and results of a task's predecessors must be transferred to the local memory before this task starts execution. For example, if a task tk_i is assigned to the CPU and its predecessor tk_j is running on the FPGA, before tk_i can start execution, the results of tk_j must be transferred from the FPGA to the local memory of the CPU.

4.2 Experimental setup

The neighborhood size for the tabu search is 10 and the tabu list length is 10. The search terminates if it cannot find a better solution after 200 iterations. In simulated annealing, the cooling rate α is 0.99, and it changes neighborhood function if current function yields no improvement after 200 iterations.

The experimental data set consists of randomly generated task graphs and task graphs of five real applications.
The following parameters are involved in generating random task graphs:

- **GS:** Graph size, it is defined as the number of tasks in a task graph.
- **ANS:** Average number of immediate successors per task in a task graph.
- **CCR:** Computation to communication ratio, which is the ratio between average processing time of tasks and average data transfer time among tasks in a task graph.
- **TA:** The target architecture used to evaluate the speedup of mapping/scheduling solutions, i.e. the fully connected or bus architecture (Section 4.1).

Directed acyclic task graphs with GS from 10 to 80, ANS from 2 to 5, and CCR from 0.1 to 100 are analyzed, 10 task graphs of each configuration are generated.

4.3 Performance evaluation

The overall processing time using a single CPU is divided by the overall processing time using the reference heterogeneous computing system to obtain a speedup:

$$speedup = \frac{overall\ processing\ time_{\sin gle\ CPU}}{overall\ processing\ time_{reference\ system}} \qquad (2)$$

Two factors: average speedup and speedup improvement are used to measure the performance of different experimental setups. They are defined as follows:

- **Average Speedup**: Each experimental setup is run for a given duration (time constraint) and the best speedup (BSU) is recorded. An average BSU of 10 runs is calculated and used as an average speedup, which represents the average quality of mapping/scheduling solutions a particular experimental setup can achieve. A higher average speedup means the solution is better.
- **Speedup Improvement**: Assume that the average speedups of two experimental setups, S1 and S2, are su1 and su2 respectively. The speedup improvement of su1 over su2 is calculated as:

$$speedup\ improvement = \frac{su1 - su2}{su2} \times 100\% \qquad (3)$$

Hence, a larger speedup improvement means the mapping/scheduling solution found by S1 is better.

TABLE 2
Average speedups of seven basic neighborhood functions. Experimental setup: target architecture = fully connected, data set = random graphs, average number of successors per task (ANS) = 2, computation to communication ratio (CCR) = 10. TABU: Tabu search, SA: simulated annealing.

Search method	Neighborhood function	Graph size (No. of tasks)			
		10	30	50	80
TABU	RanSwp	3.07	3.81	3.08	3.05
	GudSwp	3.10	4.35	3.70	3.30
	BstRlo	3.34	5.43	5.00	3.41
	RanRlo	3.33	5.35	4.99	4.43
	RanRloClu	3.34	5.59	5.23	4.75
	GudRloClu	3.34	6.45	6.82	6.81
	GudRlo	3.34	6.46	6.81	6.82
SA	RanSwp	3.04	4.90	4.17	3.65
	GudSwp	2.98	4.56	4.51	4.22
	BstRlo	3.34	5.78	5.57	5.49
	RanRlo	3.22	5.69	5.85	5.76
	RanRloClu	3.24	5.74	5.90	5.60
	GudRloClu	3.34	6.37	6.59	6.50
	GudRlo	3.34	6.39	6.58	6.56

TABLE 3
Average speedups of multiple neighborhood functions. For instance, (RanSwp,GudSwp) means using both
neighborhood functions RanSwp and GudSwp. Experimental setup: target architecture = fully connected, data set = random graphs, average number of successors per task (ANS) = 2, computation to communication ratio (CCR) = 10. TABU: tabu search, SA: simulated annealing.

Search method	Multiple neighborhood functions	Graph size (No. of tasks)			
		10	30	50	80
TABU	(RanSwp, GudSwp)	3.07	4.63	3.94	3.47
	(RanRloClu, RanSwp)	3.34	5.41	4.92	4.32
	(RanRlo, RanSwp)	3.34	5.66	4.99	4.43
	(RanRloClu, GudRloClu)	3.34	6.40	6.62	6.55
	(GudRloClu, RanSwp)	3.34	6.42	6.62	6.57
	(RanRlo, GudRloClu)	3.34	6.40	6.65	6.59
	(GudRlo, RanSwp)	3.34	6.44	6.64	6.60
	(RanRloClu, GudSwp)	3.34	6.45	6.91	7.08
	(RanRlo, GudSwp)	3.34	6.42	6.90	7.10
	(GudRlo, GudSwp)	3.34	6.50	7.00	7.13
	(GudRloClu, GudSwp)	3.34	6.51	7.04	7.17
SA	(RanSwp, GudSwp)	2.99	4.83	4.70	4.07
	(RanRloClu, GudRloClu)	3.34	6.37	6.48	6.37
	(GudRlo, RanRlo)	3.34	6.35	6.53	6.43
	(RanRlo, GudRlo)	3.33	6.34	6.52	6.45
	(RanRloClu, RanSwp)	3.29	6.21	6.52	6.45
	(GudRloClu, RanRloClu)	3.34	6.36	6.63	6.50
	(GudRlo, RanSwp)	3.34	6.43	6.67	6.59
	(RanSwp,,RanRloClu)	3.34	6.44	6.73	6.64
	(RanSwp, RanRlo)	3.34	6.44	6.69	6.64
	(RanRlo, RanSwp)	3.29	6.20	6.59	6.68
	(GudSwp, RanRloClu)	3.32	6.28	6.60	6.75
	(GudSwp, RanRlo)	3.32	6.28	6.62	6.76
	(GudRloClu, RanSwp)	3.34	6.44	6.77	6.67
	(RanRloClu, GudSwp)	3.28	6.28	6.72	6.86
	(RanRlo, GudSwp)	3.28	6.29	6.72	6.93
	(GudRlo, GudSwp)	3.34	6.46	6.84	6.98

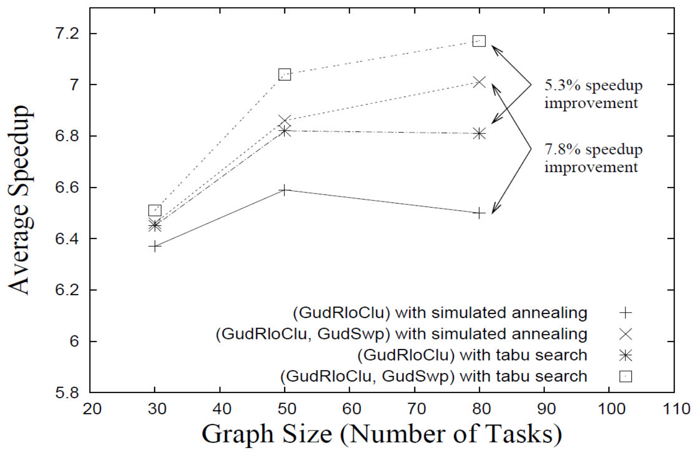

Fig. 7. Average speedup comparison between strategies with single and multiple neighborhood functions. The percentages show the speedup improvement of using multiple neighborhood functions over single neighborhood function strategy. Experimental setup: target architecture = fully connected, data set = random graphs, average number of successors per task (ANS) = 2, computation to communication ratio (CCR) = 10.

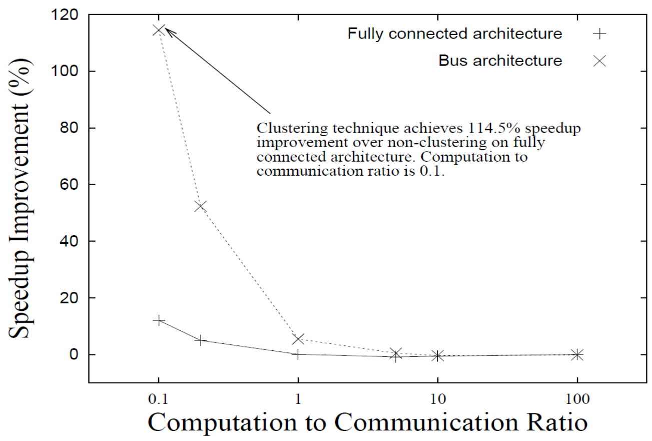

Fig. 8. Speedup improvement of the clustering technique using (GudRloClu,GudSwp) over the non-clustering approach using (GudRlo,GudSwp) for different computation to communication ratios. Experimental setup: data set = random graphs, graph size = 80 tasks, average number of successors per task (ANS) = 2.

4.4 Single and multiple neighborhood functions

Tables 2 and 3 show the average speedup obtained using integrated approach with single and multiple neighborhood functions respectively. The results of using the strategy with two neighborhood functions are designated by the notation (NFa, NFb). For simulated annealing, this notation also denotes the order to choose the neighborhood function: NFa is used first to generate neighbors, changing to NFb if there is no improvement after 200 iterations. The notation (NFa) means using a single neighborhood function NFa. Average speedups of different neighborhood functions are obtained by given the same search time constraint, in particular, the search time constraints for graph size 10, 30, 50, and 80 are 10, 30, 50, and 80 seconds respectively.

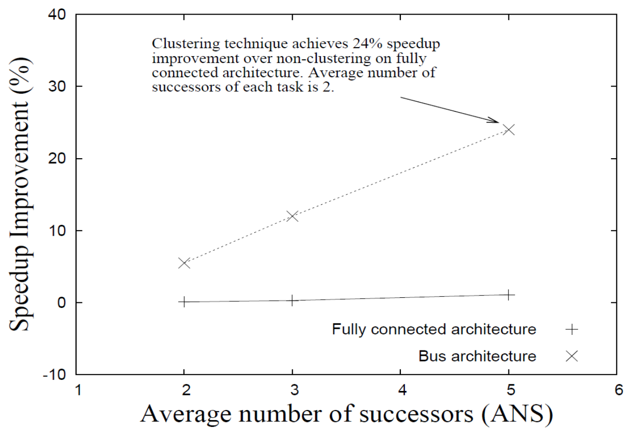

Fig. 9. Speedup improvement of the clustering technique using (GudRloClu,GudSwp) over the non-clustering approach using (GudRlo,GudSwp) for different ANSes. Experimental setup: data set = random graphs, graph size = 80 tasks, computation to communication ratio (CCR) = 1.

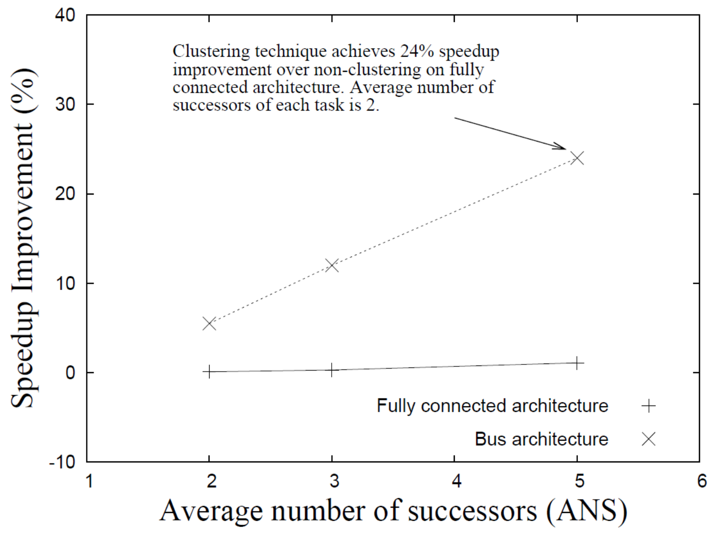

Fig. 10. Speedup improvement of this work over a separate approach in [16] for different computation to communication ratios. Experimental setup: data set = random graphs, graph size = 80 tasks, average number of successors per task = 2.

In Table 2, both results obtained using tabu search and simulated annealing shows that GudRloClu and GudRlo achieve the highest average speedup: their values are almost twice as much as others for most problem instances.
Table 3 shows the average speedups of using multiple neighborhood functions. It is found that the two neighborhood functions strategy with clustering i.e. using (GudRloClu,GudSwp), achieves highest speedup. Both tables 2 and 3 show that tabu search outperforms simulated annealing.

An average speedup comparison between single and multiple neighborhood functions is shown in Figure 7. This clearly shows that using a strategy with two neighborhood functions yields higher speedup than a single neighborhood function. The improvement is more significant for larger problem sizes. For instance, using (GudRloClu,GudSwp) with tabu search outperforms GudRloClu with tabu search by 5.3% for the case with 80 tasks, and using (GudRloClu,GudSwp) with simulated annealing outperforms GudRloClu with simulated annealing by 7.8%. Figure 7 shows that tabu search with (GudRloClu,GudSwp) achieves the highest speedup, which is actually the highest among all experimental setups chosen in this work; this strategy is used later in our experiments with real applications (Section 4.6).

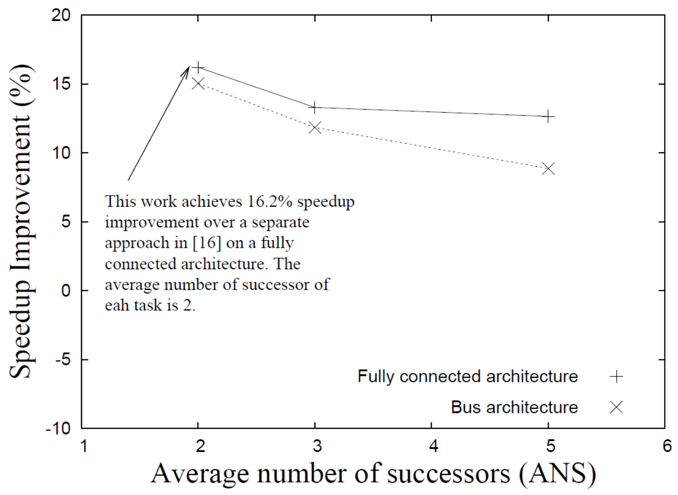

Fig. 11. Speedup improvement of this work over a separate approach in [16] for different average number of successors per task. Experimental setup: data set = random graphs, graph size = 80 tasks, computation to communication ratio = 10.

TABLE 4

Speedup improvements of this work over other two approaches using random graphs. SEP: separate approach, INT: integrating mapping and scheduling only. This work uses tabu search with neighborhood functions
(GudRloClu,GudSwp). Experimental setup: target architecture = fully connected, data set = random graphs, computation to communication ratio = 10.

Graph size (No. of task)	10	30	50	80
SEP, TABU [16]	0.9%	10.5%	13%	16.2%
INT, TABU [19]	0%	19.9%	40.8%	110.3%

Figure 8 shows the speedup improvement of the proposed clustering technique over the non-clustering approach for different computation to communication ratios. It is found that the speedup is significantly improved for high data contention cases, e.g. an improvement of 114.5% is obtained using a bus architecture with CCR equal to 0.1. On average, the clustering technique achieves higher improvement on bus architecture. The speedup improvements for different number of successors are shown in Figures 9. Bus architecture still achieves higher improvement and the improvement is higher for the cases of large number of successors which have higher data contention. These results reflect that the clustering technique is able to reduce communication overhead.

Table 4 shows the speedup improvements of this work over two other approaches: a best-relocation based integrated approach [19], and a separate approach proposed in [16] where tabu search was found to yield best performance. Our approach outperforms both approaches in all problem instances; the improvement is more significant for larger problems, it is 16.2% higher than the separate approach and 110.3% higher than the best relocation based approach for the case with 80 tasks. Figure 10 and 11 shows the speedup improvement of the proposed integrated approach over the separate approach. It is found that the integrated approach outperforms the separate approach in most of the cases. However, their performance is similar for cases with high data contention, e.g. bus architecture with CCR equals to 0.1. This is because tasks are intended to be mapped to one processing element in such a case. The separate approach which is mapping biased can thus generate similar solutions. In Figure 10, one can also see that the speedup improvement decreases for cases with lower data contention. This is because lower communication demands diminish the advantage of the clustering technique which is designed to reduce data transfer overhead.

TABLE 5
Speedup improvements of this work over other two approaches for different search time constraints using FFT. This work uses tabu search with neighborhood functions (GudRloClu,GudSwp).

Time constraint (s)	10	30	30
SEP, TABU [16]	4.3%	9.1%	10.1%
INT, TABU [19]	158.1%	153%	138.5%

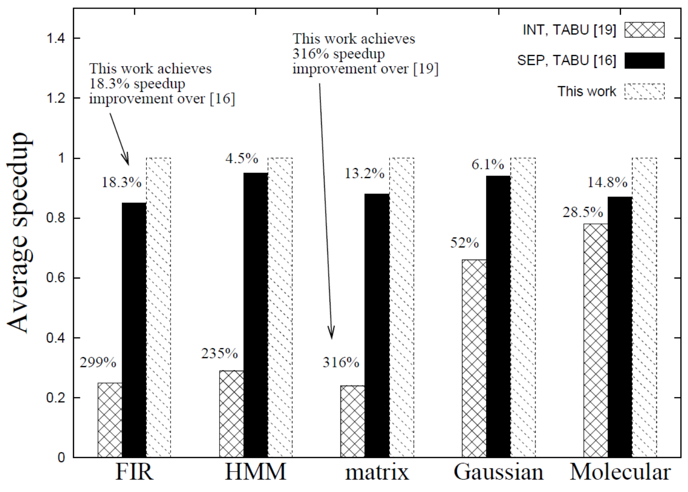

Fig. 12. Normalized average speedup for various applications given a 30-second search time constraint and the speedup improvement of this work over the other two approaches. This work uses tabu search with neighborhood functions (GudRloClu,GudSwp).

4.6 Real applications

Apart from random graphs, the commonly used butterfly structure for the fast Fourier transform (FFT) [37] is used as an additional example. A 32-point FFT containing 80 butterfly nodes with each node regarded as a single task is used. The execution time of each node is measured for processing 1.0×10^7 32-point FFTs, and the communication speed is assumed to be 1.0×10^8 data elements per second. For an Intel Pentium-4 3.2GHz microprocessor and an Atmel mAgic floating-point digital signal processor [38] the execution times of each node are 384ms and 100ms respectively. A fully pipelined architecture for the butterfly node has been developed for a Xilinx Virtex-II XC2V6000 FPGA [39] using the Haydn design-flow [40]. The execution time for this architecture is 91ms.

Table 5 illustrates the speedup improvements for different search time constraints. One can see that solutions with higher speedup can be found using an integrated approach with tabu search and (GudRloClu,GudSwp). The maximum improvement is 10.0% over the separate approach and 158.1% higher than the best relocation based approach which uses neighborhood function BstRlo.

Besides FFT, five more applications are employed to evaluate the proposed approach in this work; these include FIR filtering, hidden Markov model (HMM) decoding for pattern recognition, matrix multiplication, Gaussian elimination, and the molecular dynamics code used in [8]. Figure 12 illustrates the normalized average speedup given a 30-second search time constraint, the proposed multiple neighborhood function strategy outperforms the other two approaches in all cases, the corresponding improvements over the separate approach are 18.3%, 4.5%,

13.2%, 6.1%, and 14.8% respectively. The improvements for HMM is less significant; one reason is that the amount of data flow is smaller in these two applications, so the penalty of inappropriate task mapping using the separate approach is also less significant.

5 CONCLUSIONS

An integrated hardware/software codesign approach using multiple neighborhood function strategy is presented where clustering, mapping, and scheduling are integrated in a single step. It is found that the clustering technique outperforms a non-clustering approach and the average speedup can be further improved using the multiple neighborhood function strategy. For instance, using the clustering technique, the average speedup can be improved by up to 114.5%, and the multiple neighborhood function strategy can improve the average speedup by up to 7.8%. Experimental results obtained using both randomly generated task graphs and six real applications show that the proposed integrated approach with multiple neighborhood functions is superior to previous approaches in terms of speedup by up to 18.3%. Current and future work includes exploring opportunities for carrying out mapping and scheduling at run time, and extending the proposed approach to cover a wide range of applications and systems.

Acknowledgement The support of Macao Science and Technology Development Fund (Grant No. 058/2010/A) is gratefully acknowledged.

REFERENCES

[1] K. R. Pattipati, T. Kurien, R. T. Lee, and P. B. Luh, "On Mapping a Tracking Algorithm Onto Parallel Processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 5, pp. 774–791, 1990.

[2] T. Pop, P. Eles, and Z. Peng, "Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems," in *Proceedings of the tenth International Symposium on Hardware/software Codesign*, 2002, pp. 187–192.

[3] Y. K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[4] Y. Shin and K. Choi, "Enforcing schedulability of multi-task systems by hardware-software codesign," in *Proceedings of the Fifth International Workshop on Hardware/Software Codesign*, 1997, pp. 3–7.

[5] Y. Guo, C. Hoede, and G. Smit, "A pattern selection algorithm for multi-pattern scheduling," in *Proceedings of the Parallel and Distributed Processing Symposium*, 2006, pp. 25–29.

[6] I. Ahmad, M. K. Khodhi, and R. UI-Mustafa, "DPS: Dynamic Priority Scheduling Heuristic for Heterogeneous Computing Systems," *IEE Proceedings of Computers and Digital Techniques*, vol. 145, no. 6, pp. 411–418, 1998.

[7] G. C. Sih and E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, 1993.

[8] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[9] M. Hosseinzadeh and H. S. Shahhoseini, "Earliest Starting and Finishing Time Duplication-Based Algorithm," in *Proceeding of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2009, pp. 49–56.

[10] D. Bozdaˇg, F. ¨Ozg¨uner, and U. V. Catalyurek, "Compaction of Schedules and a Two-Stage Approach for Duplication-Based DAG Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857–871, 2009.

[11] H. Liu and D. F. Wong, "Integrated Partitioning and Scheduling for Hardware/Software Co-design," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1998, pp. 609–614.

[12] M. Y. Wu, W. Shu, and J. Gu, "Efficient Local Search for DAG Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 617–627, 2001.

[13] S. C. Kim, S. Lee, and J. Hahm, "Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1489–1502, 2007.

[14] J. I. Hidalgo and J. Lanchares, "Functional Partitioning for Hardware-Software Codesign Using Genetic Algorithms," in *Proceedings of the 23rd EUROMICRO Conference on New Frontiers of Information Technology*, 1997, pp. 631–638.

[15] S. Banerjee and N. Dutt, "Efficient Search Space Exploration for HW-SW Partitioning," in *Proceedings of the International Symposium on Hardware/software Codesign and System Synthesis*, 2004, pp. 122–127.

[16] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Hardware/Software Codesign: A Systematic Approach Targeting Data-intesive Applications," *IEEE Signal Processing Magazine*, vol. 22, no. 3, pp. 14–22, May 2005.

[17] H. Yu, "A Hybrid GA-based Scheduling Algorithm for Heterogeneous Computing Environments," in *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling*, 2007, pp. 87–92.

[18] L. Shang, R. P. Dick, and N. K. Jha, "SLOPES: Hardware-Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs," *IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 508–526, 2007.

[19] S. C. S. Porto and C. C. Ribeiro, "A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints," *International Journal of High-Speed Computing*, vol. 7, pp. 45–71, 1995.

[20] Y. M. Lam, J. G. F. Coutinho, W. Luk, and P. H. W. Leong, "Integrated Hardware/Software Codesign for Heterogeneous Computing Systems," in *Proceedings of the IEEE IV Southern Programmable Logic Conference*, 2008, pp. 217–220.

[21] H. Barada, S. M. Sait, and N. Baig, "Task Matching and Scheduling in Heterogeneous Systems Using Simulated Evolution," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, 2001, pp. 875–882.

[22] Y. W. Wong, R. S. M. Goh, S. H. Kuo, and M. Y. H. Low, "A Tabu Search for the Heterogeneous DAG Scheduling Problem," in *Proceeding of the International Conference on Parallel and Distributed Systems*, 2009, pp. 663–670.

[23] S. Gupta and G. Agarwal, "Task Scheduling in Multiprocessor System Using Genetic Algorithm," in *Proceeding of the Second International Conference on Machine Learning and Computing*, 2010, pp. 267–271.

[24] A. Bednarski and C. Kessler, "Integer Linear Programming versus Dynamic Programming for Optimal Integrated VLIW Code Generation," in *Proceedings of the 12th International Workshop on Compilers for Parallel Computers*, 2006, pp. 73–85.

[25] S. O. M. F. Redaelli, M. D. Santambrogio, "An ILP formulation for the Task Graph Scheduling Problem tailored to bi-dimensional Reconfigurable," in *Proceeding of the International Conference on Reconfigurable Computing and FPGAs*, 2008, pp. 97–102.

[26] R. Cordone, F. Redaelli, M. A. Redaelli, M. D. Santambrogio, and D. Sciuto, "Partitioning and Scheduling of Task Graphs on Partially Dynamically Reconfigurable FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 662–675, 2009.

[27] M. M. Rahman and M. F. I. Chowdhury, "Examining Branch and Bound Strategy on Multiprocessor Task Scheduling," in *Proceeding of the International Conference on Computer and Information Technology*, 2009, pp. 162–167.

[28] L. Yang, T. Gohad, P. Ghosh, D. Sinha, A. Sen, and A. Richa, "Resource Mapping and Scheduling for Heterogeneous Network Processor Systems," in *Proceedings of the IEEE International Symposium on Architecture for networking and communications systems*, 2005, pp. 19–28.

[29] J. Cong, K. Gururaj, and G. Han, "Synthesis of Reconfigurable High-performance Multicore Systems," in *Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2009, pp. 201–208.

[30] T. D. Braun, H. J. Siegel, N. Beck, L. L. B¨ol ¨oni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computering*, vol. 61, pp. 810–837, 2001.

[31] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign," *Journal on Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, 2002.

[32] T. C. Hu, "Parallel Sequencing and Assembly Line Problems," *Operations Research*, vol. 9, no. 6, pp. 841–848, 1961.

[33] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Inc, 1998.

[34] C. R. Reeves, "Modern Heuristic Techniques," in *Modern Heuristic Search Methods*, V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, Eds. John Wiley & Sons Ltd, 1996, pp. 1–25.

[35] E. Aarts and J. K. Lenstra, *Local Search in Combinatorial Optimization*. John Wiley & Sons Ltd, 1997.

[36] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*. Pitman publishing, 1989.

[37] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*. The McGraw-Hill Companies, Inc, 2002.

[38] *Atmel DIOPSIS 740 Dual-core DSP datasheet*, Atmel Corporation, 2004.

[39] *Virtex-II Platform FPGAs: Complete Data Sheet*, Xilinx Inc., 2005.

[40] J. G. F. Coutinho, J. Jiang, andW. Luk, "Interleaving Behavioral and Cycle-Accurate Descriptions for Reconfigurable Hardware Compilation," in *Proc. of the IEEE Symposium on FCCM*, 2005, pp. 245–254.