Optimal Functional Unit Assignment and Voltage Selection for Pipelined MPSoC with Guaranteed Probability on Time Performance

Weiwen Jiang Edwin H.-M. Sha Qingfeng Zhuge Hailiang Dong Xianzhang Chen
College of Computer Science, Chongqing University, Chongqing, 400044, China
{jiang.wwen, edwinsha, qfzhuge, leondong1993 xzchen109}@gmail.com

Abstract

Pipelined heterogeneous multiprocessor system-on-chip (MPSoC) can provide high throughput for streaming applications. In the design of such systems, time performance and system cost are the most concerning issues. By analyzing runtime behaviors of benchmarks in real-world platforms, we find that execution times of tasks are not fixed but spread with probabilities. In terms of this feature, we model execution times of tasks as random variables. In this paper, we study how to design high-performance and low-cost MPSoC systems to execute a set of such tasks with data dependencies in a pipelined fashion. Our objective is to obtain the optimal functional unit assignment and voltage selection for the pipelined MPSoC systems, such that the system cost is minimized while timing constraints can be met with a given guaranteed probability. For each required probability, our proposed algorithm can efficiently obtain the optimal solution. Experiments show that other existing algorithms cannot find feasible solutions in most cases, but ours can. Even for those solutions that other algorithms can obtain, ours can reach 30% reductions in total cost compared with others.

CCS Concepts •**Computer systems organization** → **System on a chip**; **Embedded systems**

Keywords Functional unit assignment, application-specific system, energy efficiency, probabilistic execution time, optimal algorithms

1. Introduction

With the growing demand for high-performance and low-cost design, state-of-the-art embedded devices (e.g., mobile platforms) move toward heterogeneous Multiprocessor System-on-Chip (MPSoC) platforms. On such platforms, the increasing use of streaming applications, which are executed in a pipelined fashion, promotes the rise of a specialized MPSoC, called pipelined MPSoC (Javaid et al. 2010, 2014). A pipelined MPSoC is organized in a pipelined configuration, in which each core repeatedly executes a pipeline stage. This paper focuses on efficiently constructing pipelined heterogeneous MPSoCs for streaming applications.

In the design of pipelined heterogeneous MPSoCs, it is critical to deal with the conflicting needs of low cost and high time performance. The cost can be related to price, energy, reliability, etc. Given an application, the problem is how to construct a system having the minimum total system cost while satisfying timing constraints. This problem involves three decision functions. *Partition:* group tasks into pipeline stages. *Assignment:* assign pipeline stages to heterogeneous functional units (or called cores). *Selection:* select running voltage/frequency for each functional unit. This paper will propose algorithms to obtain the global optimal partition, assignment, and selection.

There are two facts that make the optimization problem challenging. *First*, the execution time of a task may not be fixed. The uncertainties within execution times make it complicated to ensure that the constructed systems can satisfy the required time performance. *Second*, the determinations on partitions, assignments, and selections are not independent but coupled to each other. In other words, these functions should be considered simultaneously, which drastically expands the design space.

For the *first challenge*, we find the varying execution times of a task are spread in a wide range on real-world mobile platforms. Based on our observation, we model the probabilistic execution times of tasks as random variables. In consequence, the determinations of partitions, assignments, and selections need to be conducted in probabilistic scenarios. For the resultant pipeline, its throughput is also a random variable. Given a set of dependent tasks with probabilistic execution times, our objective is to minimize the total system cost under a required throughput with a given guaranteed probability.

For the *second challenge*, it requires efficient algorithms to explore design space. In existing research efforts, the optimization approaches can be classified into two main categories. The *first category* includes heuristics: (1) greedy approaches, such as optimizing different functions in multiple phases (Niu et al. 2014); (2) stochastic search algorithms, such as particle swarm optimizations (Salman et al. 2002) and genetic algorithms (Wang et al. 2011). These approaches, however, cannot guarantee the optimality of the obtained solutions. The *second category* includes Integer Linear Programming (ILP) formulations (Kuang et al. 2005). But ILP suffers from the large computational complexity. This paper does not apply the above approaches. We devise efficient algorithms based on the dynamic programming approach, which can obtain the global optimal solution by the consideration of multiple decision functions simultaneously.

LCTES'17, June 21–22, 2017, Barcelona, Spain
© 2017 ACM. 978-1-4503-5030-3/17/06...$15.00
http://dx.doi.org/10.1145/3078633.3081036

The main contributions of this paper are listed as follows.

- By analyzing benchmarks on real-world platforms, we find that the execution time of a task may not be fixed but widely spread in probabilistic distributions. Based on our observations, we model the execution times as random variables.
- We propose efficient algorithms to construct pipelined heterogeneous MPSoC in probabilistic scenarios. The resultant systems can achieve the minimum system cost while satisfying timing constraints with a given guaranteed probability.
- We conduct comprehensive experiments on the state-of-the-art mobile platforms with real benchmarks to minimize system energy consumptions. The results demonstrate the effectiveness and practicability of our proposed algorithms.

We conduct a set of experiments on a real mobile platforms, Odroid-XU3 (Odroid-XU3), on which a set of benchmarks from StreamIT (Thies et al. 2002) is performed to demonstrate the energy consumption savings achieved by our propose algorithms over existing approaches. Specifically, compared with ILP using worst-case scenarios, we can achieve 31.33% and 21.73% reductions in energy consumption when the required guaranteed confidence probabilities are 90% and 99%, respectively. Compared with two heuristics, (Niu et al. 2014) and (Wang et al. 2011), using probabilistic scenarios, average energy consumption savings obtained by our algorithms are 23.93% and 13.91%, respectively. As for the running time, our algorithms only take seconds to obtain the optimal solutions. In contrast, the ILP using worst-case scenarios and existing heuristics using probabilistic scenarios take minutes even hours to obtain feasible solutions.

The rest of the paper is organized as follows. Section 2 introduces the system models and the problem definition. Section 3 gives motivational examples to show the properties of probabilistic execution times on tasks and the significance to conduct optimizations in probabilistic scenarios. Proposed algorithms are presented in Section 4. And experimental results are given in 5. Section 6 provides the literature review. Finally, in Section 7, we conclude this paper and discuss future work.

2. Models and Problem Definition

In this section, we will first define the application and architecture models. Then, we will introduce the functional unit assignment. Next, we discuss how to compute the system total cost and system performance. Based on these understandings, we finally give the problem definition.

2.1 Application Model

This paper considers the applications which are executed iteratively on a set of input data (e.g. streaming applications and convolutional neural network applications). We use task graph to model an application. A task graph $G = \langle V, E \rangle$ consists of a set of nodes V representing tasks, and a set of edges $E \subseteq V \times V$ representing data dependencies. For instance, the task graph of JPEG Encoder (`jpeg`) is shown in Figure 1(a), which contains 17 nodes.

Due to limited computational resources, we need to conduct partition on task graphs. A partition of a task graph G is to put nodes into one pipeline stage. Define the partition $P(G)$ to be a set of pipeline stages in G. And $p_i \in P(G)$ is the i^{th} pipeline stage, which contains a set of tasks. As shown in Figure 1(a), we have $p_1 = \{R, C, F\}$.

2.2 Architecture Model

In a pipelined MPSoC, cores are organized in a pipeline configuration, where each core corresponds to a pipeline stage. If two pipeline stages have data dependencies, it typically employs FIFO buffers between cores for communication (Javaid et al. 2010, 2014). We call that pipeline stages p_i and p_j have data dependency, if $\exists v_i \in p_i$ and $\exists v_j \in p_j$, such that there is an edge from v_i to v_j. For instance, the pipelined MPSoC shown in Figure 1(b) is constructed from Figure 1(a). The constructed system can be executed in a pipelined fashion. Specifically, for two consecutive pipeline stages, p_6 and p_7, if p_7 executes at its i^{th} iteration, then p_6 executes its $(i+1)^{th}$ iteration simultaneously. Note that since p_2 and p_3 have no data dependencies, they can be executed in parallel.

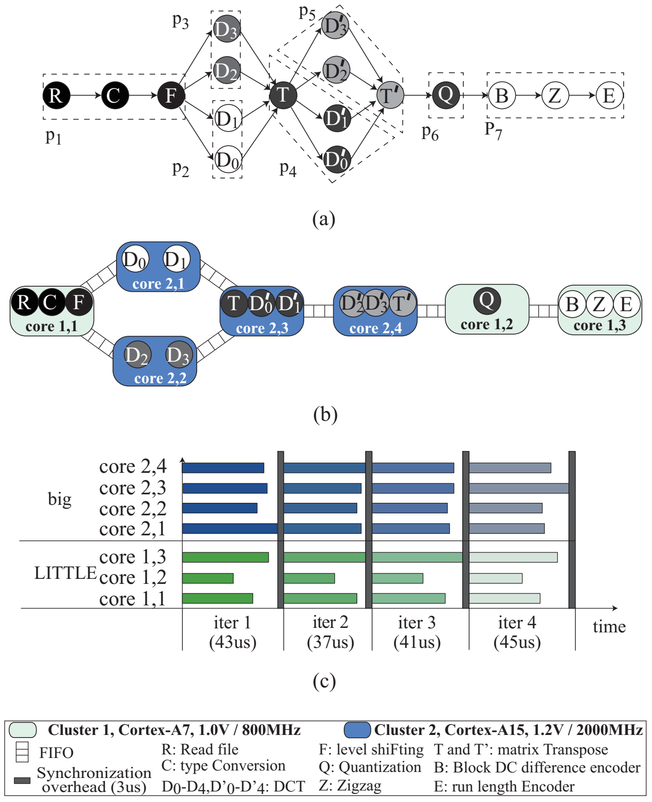

Figure 1. System models: (a) task graph of `jpeg` with partitions; (b) pipelined heterogeneous MPSoC with assignments; (c) schedule graph in steady state.

In this work, we consider heterogeneous MPSoC platforms. An example of commercial heterogeneous MPSoC is ARM big.LITTLE architecture which integrates a cluster of high-performance ARM Cortex-A15 (big) cores and a cluster of low-power ARM Cortex-A7 (little) cores. It provides multiple voltage/frequency (abbreviate as "V/F") levels to save energy consumption. All cores in the same cluster are set to the same V/F level.

A heterogeneous MPSoC is defined as $H = \langle C, L, m \rangle$. It consists of a set of clusters C. For cluster $c_i \in C$, the attribute $m(c_i)$ represents the number of cores in c_i. Note that when $m(c_i)$ equals 1, it indicates each core can execute at its own V/F level. L_i is a set of V/F levels at which c_i can operate. $L_i = \{(V_{dd_{i,1}}, f_{i,1}), \cdots, (V_{dd_{i,k}}, f_{i,k})\}$ indicates that c_i can operate at k different levels; $l_{i,j} \in L_i$ represents the j^{th} level of c_i (i.e., voltage level $V_{dd_{i,j}}$ and frequency $f_{i,j}$). Table 1 illustrates six V/F levels for two clusters in Odroid-XU3, a state-of-the-art mobile platform.

A V/F selection of heterogeneous MPSoC $H = \langle C, L, m \rangle$ is to select a V/F level for each cluster. Define the V/F selection S to be a function from domain C to L. For cluster $c_i \in C$, $S(c_i) = l_{i,j}$

Table 1. Six V/F levels in Odroid-XU3 (Odroid-XU3)

Notation		$l_{1,1}$	$l_{1,2}$	$l_{1,3}$	$l_{1,4}$	$l_{1,5}$	$l_{1,6}$
Cortex	freq (MHz)	400	600	800	1000	1200	1400
A7	vol (mV)	916.25	917.5	992.5	1066.25	1141.25	1240

Notation		$l_{2,1}$	$l_{2,2}$	$l_{2,3}$	$l_{2,4}$	$l_{2,5}$	$l_{2,6}$
Cortex	freq (MHz)	800	1200	1400	1600	1800	2000
A15	vol (mV)	910	983.75	1012.5	1076.25	1152.5	1278.75

Table 2. Probabilistic execution times of DCT

	Cortex-A7 (c_1)				**Cortex-A15 (c_2)**			
	L_1	time	prob	cost	L_2	time	prob	cost
$\boldsymbol{D_0}$	$l_{1,1}$	62us	92.24%	3.17uJ	$l_{2,1}$	33us	95.85%	9.57uJ
$\boldsymbol{D_1}$		125us	7.76%			64us	4.15%	
$\boldsymbol{D_2}$	$l_{1,5}$	27us	92.84%	5.48uJ	$l_{2,5}$	13us	98.01%	18.55uJ
$\boldsymbol{D_3}$		53us	7.16%			26us	1.99%	

indicates that $(V_{dd_{i,j}}, f_{i,j})$ is selected for c_i. We use the notation $\boldsymbol{S(H)}$ to represent the V/F selection of H; i.e. $\forall c_i \in C$, $S(c_i)$.

2.3 Functional Unit Assignment

Due to the heterogeneity in architectures, cores in different clusters can execute the same pipeline stage using varying execution times with different cost which may be any kinds of cost, such as price, energy consumption and reliability cost.

For a cluster c_i in an MPSoC, it has two attributes. (1) Attribute $et_{c_i}(v, l_{i,j})$ represents the execution time of node $v \in V$ on cluster c_i at V/F level $l_{i,j}$. Note that the execution time of a node is not fixed (see Section 3). Based on the properties found in this paper, we model execution times as random variables. (2) Attribute $ec_{c_i}(v, l_{i,j})$ represents the execution cost. For example, if the execution cost represents the expected energy consumption, it can be obtained by the production of power and the expected execution time (Qiu et al. 2006).

As shown in Table 2, execution times of DCT tasks on cluster c_2 with level $l_{2,1}$ can be represented by $et_{c_2}(D_0, l_{2,1}) = [(33, 95.85\%), (64, 4.15\%)]$. It means that DCT task will be finished in 33us with probability 95.85%, and between 33us and 64us with probability 4.15%. Hence, the expected time is 34.29us. The measured power of cores in c_2 for DCT task is 0.1644W. Thus, the expected energy is 9.57uJ.

An assignment of a task graph G is to assign each pipeline stage to a cluster. Define an assignment A to be a function from domain $P(G)$ to C. For a pipeline stage p_i and a cluster $c_j \in C$, $A(p_i) = c_j$ indicates that p_i will be executed by a core in cluster c_j. Note that since each core corresponds to a pipeline stage, we can assign $m(c_j)$ stages to c_j at most. We use notation $\boldsymbol{A(G)}$ to represent the assignment of G under partition $P(G)$; i.e. $\forall p_i \in P(G)$, $A(p_i)$.

2.4 System Cost and Performance

Total execution cost. Given an MPSoC H and a task graph G, we define the system total execution cost under selection $S(H)$, partition $P(G)$ and assignment $A(G)$, denoted by $EC_{S,P,A}(H, G)$, to be the summation of the costs (e.g., energy consumption) of all nodes in the given task graph G. We can calculate total cost $EC_{S,P,A}(H,G) = \sum_{\forall p_i \in P(G)} \sum_{\forall v \in p_i} \left\{ec_{A(p_i)}(v, S(A(p_i)))\right\}$.

Notice that the algorithms presented in this paper can solve any kinds of cost. In the following of this paper, we take the expected energy consumption as an example, which should be minimized to achieve energy efficiency.

Iteration period. Given an MPSoC H and a task graph G, notation $IP_{S,P,A}(H, G)$ stands for the iteration period (i.e., reciprocal of throughput) of the pipeline under selection $S(H)$, partition $P(G)$, and assignment $A(G)$. Iteration period can be obtained by maximizing latencies of all pipeline stages, where the latency of a pipeline stage is the summation of execution times of nodes in it (including the synchronization overhead δ); i.e., $IP_{S,P,A}(H,G) = \max_{\forall p_i \in P(G)} \left\{\sum_{\forall v \in p_i} et_{A(p_i)}(v, S(A(p_i))) + \delta\right\}$. Since execution times of tasks are random variables, $IP_{S,P,A}(H, G)$ is also a random variable. As shown in Figure 1(c), iteration periods in 4 contiguous iterations vary from 37us to 45us.

2.5 Problem Definition

Our objective is to construct a pipelined MPSoCs that have *the minimum total execution cost* (***EC***) while *satisfying the required iteration period* (***R***) under *a guaranteed probability* (***GP***).

The problem is defined as follows: Given an MPSoC (H), a task graph (G), a required iteration period (R) and a guaranteed probability (GP), the problem is to determine,

$\boldsymbol{S}$: V/F selections of clusters in H;

$\boldsymbol{P}$: partitions of task graph G;

$\boldsymbol{A}$: assignments of task graph G;

such that the resultant pipeline has the minimum total system cost under required iteration period R with guaranteed probability GP; i.e., $min_{\forall S, \forall P, \forall A}(EC_{S,P,A}(H, G))$, s.t., $Pr\{IP_{S,P,A}(H, G) \leq R\} \geq GP$, where Pr represents probability. Figure 1(b) shows the optimal solution for `jpeg` on Odroid-XU3 platform when $R = 45$ and $GP = 95\%$.

Note that when $GP = 100\%$, the obtained solutions can be used for hard real-time systems. It is identical to construct pipelines in "worst-case scenarios". On the contrary, when $GP < 100\%$, pipelines are constructed in "probabilistic scenarios", which can be used for soft real-time systems.

3. Motivation

This section will first present the features of execution time of benchmarks on real-world platforms. Then, we will show the importance to minimize total execution costs of pipelined heterogeneous MPSoCs in probabilistic scenarios.

In realistic execution environment, the execution times of tasks are usually not fixed due to the uncertainties in its execution, such as the effects of conditional instructions, caches, fluctuant I/O loads, system interruptions, and thread synchronization. In order to understand the properties on the probabilistic execution times, we trace the execution times of tasks in 8 benchmarks from StreamIT (Thies et al. 2002). All benchmarks are executed by Cortex-A15 cores in Odroid-XU3, as shown in Figure 2 (a). In motivational examples, we configure the V/F level of Cortex-A15 cores as 0.9V/800MHz. For each benchmark, we select one representative task. The probabilistic distributions on the execution times of tasks are shown in Figures 2(b) to (i).

From the above results, we have observed two features on the execution times of tasks on real-world platforms.

1. ***Varying execution times of a task are distributed in a wide range in real-world platforms***.

   For example, consider the DCT task in JPEG benchmark in Figure 2 (e). As shown in this figure, the DCT task can be finished in 35 time units with 95.85% probability. However, the latency that can guarantee the completion of DCT task is 64 time units. Namely, the worst-case execution time is 64. This feature indicates that using fix-time scenarios (such as worst-case execution time) will lead to inferior designs.

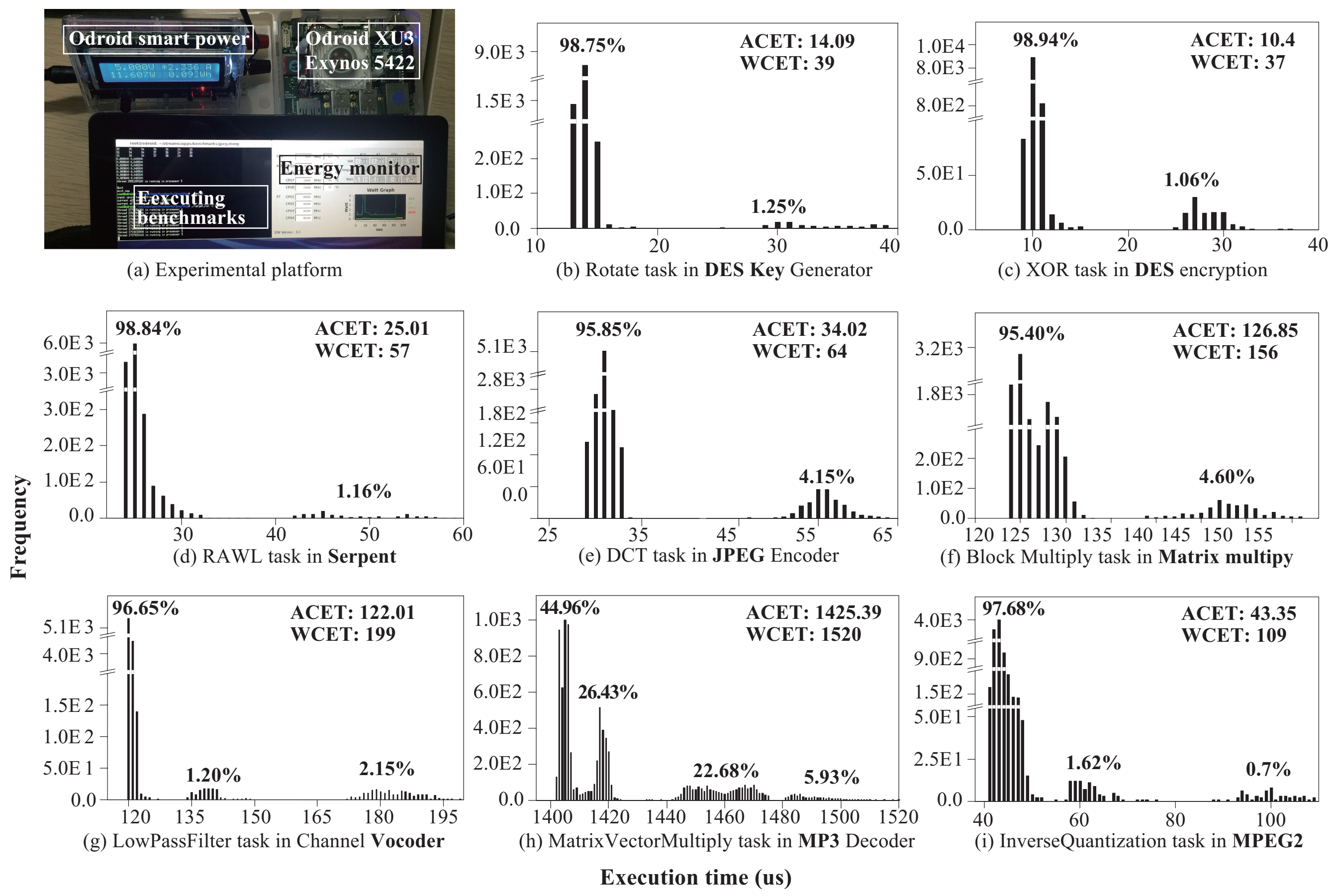

Figure 2. The experimental platform and histograms of execution times of tasks in 8 benchmarks from StreamIT (Thies et al. 2002): (a) the Cortex-A15 cores on the Odroid-XU3 platform (Odroid-XU3); (b) `deskey`; (c) `des`; (d) `serpent`; (e) `jpeg`; (f) `matmul`; (g) `vocoder`; (h) `mp3`; (i) `mpeg2`.

2. ***Varying execution times of a task are gathered in clusters***.

   For the DCT task in Figure 2 (e), its execution time rarely exists in the range from 35 to 50. There are many possible reasons to contribute to this feature, such as the interference of operating systems, the branch conditions in the tasks, etc. Based on this feature, we model execution times as discrete random variables. For instance, based on Figure 2(d), the execution time of DCT on Cortex-A15 at 0.9V/800MHz is modeled as a discrete random variable (see column c_2, row $l_{2,1}$ in Table 2).

To compare total energy consumption in fix-time scenarios and probabilistic scenarios, we construct pipelines for all benchmarks using different approaches. For fix-time scenarios, we use both Average-Case (expectation) Execution Time (ACET) and Worst-Case Execution Time (WCET). As for probabilistic scenarios, we use the Probabilistic Execution Time (PrET) modeled by random variables. Results in Table 3 are obtained by our algorithms presented in Section 4.

Table 3. Energy consumption (uJ) of 8 benchmarks

Bench.	R	ACET		WCET	**PrET**	**PrET**
	(us)	Eng.	Prob.	Eng.	$GP = 90\%$	$GP = 99\%$
`deskey`	22	13.88	45.32%	17.81	**13.88**	**14.09**
`des`	40	38.28	78.98%	71.02	**38.28**	**41.05**
`serpent`	40	37.94	74.53%	62.21	**40.51**	**44.58**
`jpeg`	55	47.46	50.36%	125.86	**79.55**	**89.66**
`matmul`	89	103.53	75.26%	147.58	**109.00**	**139.23**
`vocoder`	800	1046.23	64.45%	1631.33	**1070.52**	**1139.41**
`mp3`	2360	1673.15	47.01%	×	**1713.05**	×
`mpeg2`	155	338.81	67.48%	×	**344.68**	×

From Table 3, we have two observations. (1) Using ACET cannot guarantee the probability to achieve the required time performance. In particular, none of systems constructed by ACET can achieve more than 80% guaranteed probabilities on the required iteration period (R). The system overall performance is determined by all pipeline stages rather than the critical one. This is the root cause that ACET cannot guarantee a high probability. For instance, assuming that each pipeline stage has exactly 90% of probability to satisfy the timing constraint. Considering there are 10 pipeline stages, as a whole, the probability that the resultant system can satisfy the timing constraint is $(90\%)^{10}$, which is merely 34.87%.

(2) Compared with ACET, using WCET can guarantee the resultant systems satisfy the required probability. But it is too conservative. In conducting assignment using WCET, it requires higher frequency to meet timing constraints. By this way, extra slacks in pipeline stages are incurred, which lead to consume more energy or even worse without feasible solutions (denoted "×"). Specifically,

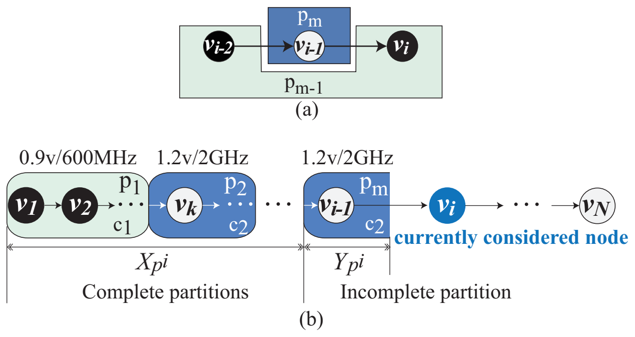

Figure 3. Examples: (a) An invalid partition; (b) S^i, P^i, A^i in the consideration of node v_i.

optimal solutions using PrET under $GP = 90\%$ achieve 32.71% reductions on average compared with WCET.

The above results demonstrate the importance of constructing pipelined heterogeneous MPSoCs using probabilistic execution time. It motivates us to find the optimal solutions in probabilistic scenarios.

4. Optimal Algorithms

This section will present efficient algorithms to construct pipelined heterogeneous MPSoCs having the minimum energy consumption while satisfying timing constraints with a guaranteed probability. There are two challenges to obtain the optimal solution. (1) *Three decision functions, $\boldsymbol{S}$, $\boldsymbol{P}$, $\boldsymbol{A}$, have to be determined at the same time.* In this paper, we simultaneously consider these functions in dynamic programming approaches. (2) *It is difficult to conduct dynamic programming approaches in probabilistic scenarios.* Specifically, instead of one local optimal solution for a subproblem in fix-time scenarios, there exists a set of possible solutions that may contribute to the global optimal solution in probabilistic scenarios. In the following, we first introduce the algorithms in fix-time scenarios. Then, we introduce how to make it work in probabilistic scenarios.

4.1 Fix-Time Scenarios

Let the topological order of a task graph be $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$. In dynamic programming algorithms, we consider these nodes one by one. When v_i is considered, tuple $\langle S^i, P^i, A^i\rangle$ represents the V/F selection $\boldsymbol{S^i}$, partition $\boldsymbol{P^i}$, and assignment $\boldsymbol{A^i}$ for the subgraph from v_1 to v_i.

Before presenting our algorithms, we introduce two useful properties. The first property is that the sequence of parallel nodes in topological order will not affect the final results. We call the nodes, between a splitter and a joiner, having no data dependencies as parallel nodes (e.g., D_0, D_1, D_2, D_3 between S and T in Figure 1(a)). We have this property because parallel nodes in streaming applications commonly have the identical behavior and execution time. The second property is that systems cannot be executed in pipelined fashion if two partitions have data dependencies between each other (see Figure 3(a)).

In terms of the first property, we can consider task graphs based on the topological order. According to the second property, if there are m partitions before v_i, then v_i can only be added into the m^{th} partition or a newly created partition. As shown in Figure 3(b), when we consider v_i, the first $m-1$ partitions are determined, called "complete partitions" (denoted by $\boldsymbol{X_{p^i}}$). But p_m can absorb new nodes, called "incomplete partition" (denoted by $\boldsymbol{Y_{p^i}}$).

In the dynamic programming algorithm, for simplicity, we consider the platform with two clusters (c_1 and c_2). We will construct

Algorithm 1 The computation of $\boldsymbol{D_{i,j,k,e,l_1,l_2,m_1,m_2}}$ in fix-time scenarios

```
Input: The computed dynamic programming cells, from node $v_1$
    to $v_i$; the required iteration period $R$; the considering node $v_i$.
Output: The dynamic programming cell, $D_{i,j,k,e,l_1,l_2,m_1,m_2}$.
 1: $PossibleSolutionSet = \emptyset$;
    // Case 1: Add $v_i$ into the former partition.
 2: $t^{i-1}_{S,P,A} = D_{i-1,j,k,e-ec_{c_j}(v_i,l_j),l_1,l_2,m_1,m_2}$;
 3: if $t^{i-1}_{S,P,A} + et_{c_j}(v_i,l_j) \leq R$:
 4:     Insert $\boldsymbol{\delta + et_{c_j}(v_i,l_j)}$ into $PosSolSet$
    // Case 2: Add $v_i$ into a new partition.
 5: $\forall c_{j^*} \in C$:
 6:     $t^{i-1}_{S,P,A} = D_{i-1,j^*,k-1,e-ec_{c_j}(v_i,l_j),l_1,l_2,m_1\text{-}\mathbf{1},m_2}$
 7:     if $t^{i-1}_{S,P,A} \neq \infty$:
 8:         $Insert\ \boldsymbol{\delta + et_{c_j}(v_i,l_j)}\ into\ PosSolSet$;
    // Finally, select the optimal (superior) solution(s).
 9: if $PossibleSolutionSet = \emptyset$:
10:     return $\infty$;
11: else
12:     return $min\{PosSolSet\}$;
```

the dynamic programming table $D_{i,j,k,e,l_1,l_2,m_1,m_2}$. Each entry records ***the minimum latency of the incomplete partition*** of all possible $\langle S^i, P^i, A^i\rangle$ tuples under the following conditions: (1) latencies of all partitions are less than or equal to the required iteration period $\boldsymbol{R}$, (2) node v_i is assigned to the cluster c_j, (3) node v_i is partitioned into the k^{th} stage, (4) the summation of energy consumption from v_1 to v_i is e, (5) the selected V/F levels are l_1 and l_2 for clusters c_1 and c_2, respectively, and (6) the number of assigned cores in clusters c_1 and c_2 are m_1 and m_2, respectively. Note that if there are no solutions satisfying the above conditions, we have $D_{i,j,k,e,l_1,l_2,m_1,m_2} = \infty$.

Then, it is clear that if $D_{N,j,k,e,l_1,l_2,m_1,m_2}$ is not ∞, there exists at least one feasible solution whose total energy consumption equaling e while the system can satisfy timing constraints. Thus, the solution with the minimum energy consumption can be obtained by finding the smallest e, such that $D_{N,j,k,e,l_1,l_2,m_1,m_2} \neq \infty$.

Now we introduce how to construct the dynamic programming table $D_{i,j,k,e,l_1,l_2,m_1,m_2}$. In every step, one more node will be considered. When we consider node v_i, there are two cases: (1) add v_i into the former partition; (2) add v_i into a new partition. In the algorithm, we use notation $t^i_{S,P,A}$ to represent the latency of the incomplete partition under tuple $\langle S^i, P^i, A^i\rangle$.

Based on the above definitions, the computation of one cell ($D_{i,j,k,e,l_1,l_2,m_1,m_2}$) in dynamic programming table is given in Algorithm 1. Specifically, the computation reflects the situation that node v_i is grouped into the k^{th} stage which is assigned to cluster c_j, and the energy consumption for the subgraph from v_1 to v_i is exactly e.

The algorithm consists two cases, for ***Case 1***, v_i and v_{i-1} are in the same partition, which indicates that v_{i-1} is assigned to cluster c_j. Total energy consumption from v_1 to v_i is exactly e, which indicates that energy consumption from v_1 to v_{i-1} is $e - ec_{c_j}(v_i, l_j)$. Hence, we obtain the time stored in the computed cell $D_{i-1,j,k,e-ec_{c_j}(v_i,l_j),l_1,l_2,m_1,m_2}$ for further consideration (line 2). In line 3, we check whether the assignment can satisfy the timing constraint or not. If $A(v_i) = c_j$ is a feasible assignment, we temporarily store it in a set, called "$PosSolSet$" (line 4). Note that we can only use a variable to store the possible solution in fix-time scenario. However, in the purpose of keeping consistency with the algorithms for probabilistic scenarios, we use a temporary set $PosSolSet$ to store possible solutions.

For ***Case 2***, we consider to add v_i into a new partition. In other words, nodes v_1 to v_{i-1} are grouped into $k-1$ complete stages. Therefore, the computation of $D_{i,j,k,e,l_1,l_2,m_1,m_2}$ is based on the cells whose third dimension, representing the number of complete stages, exactly equals $k-1$. Similarly to case 1, the fourth dimension, reflecting energy cost, should equal to $e - ec_{c_j}(v_i, l_j)$. Note that in line 6, we use $m_1 - 1$ because we assume $j = 1$, if $j = 2$, the equation in line 6 should be changed to $t^{i-1}_{S,P,A} = D_{i-1,j^*,k-1,e-ec_{c_j}(v_i,l_j),l_1,l_2,m_1,m_2\text{-}1}$. In lines 7, we check whether there exists feasible solutions. If there exists one solution, in line 8, then we set the latency of the incomplete partition as the sum of synchronization overhead δ and the execution time of task v_i.

Finally, in lines 9 to 12, we return the computed value which is the minimum execution time for all possible solutions, satisfying all conditions related to the dynamic programming cell $D_{i,j,k,e,l_1,l_2,m_1,m_2}$. The time complexity of the proposed algorithm will be discussed later.

4.2 Probabilistic Scenarios

In probabilistic scenarios, the optimization is much harder than that conducted using fix-time scenarios. In fix-time scenarios, the constructed pipelines can satisfy timing constraints by guaranteeing that the latency of the incomplete partition is less than or equal to the required iteration period in every step. But, in probabilistic scenarios, the probability on iteration period $Pr\{IP^i_{S,P,A}(H,G) \le R\}$ is contributed by every partition. For instance, let $p_1, p_2, \cdots, p_N$ be N partitions of G, and $t(p_k)$ be the latency of k^{th} partition, which is a random variable. Then $Pr\{IP^i_{S,P,A}(H,G) \le R\} = Pr\{t(p_1) \le R\} \cdot Pr\{t(p_2) \le R\} \cdots Pr\{t(p_N) \le R\}$.

Therefore, to obtain the optimal solutions in probabilistic scenarios, we need to record two kinds of information when considering v_i for each $\langle S^i, P^i, A^i\rangle$ tuple: (1) the product of probabilities of all complete partitions, $\Pi_{\forall p \in X_{P^i}} Pr\{t(p) \le R\}$, denoted by $prob^i_{S,P,A}$; (2) the latency of the incomplete partition, $t(Y_{P^i})$, denoted by $t^i_{S,P,A}$. When we construct the dynamic programming table, a lot of $(prob^i_{S,P,A}, t^i_{S,P,A})$ pairs are redundant (dominated by others).

Before showing how to remove redundant pairs, we first introduce the comparison between two random variables, $t^i_{S_0,P_0,A_0}$ and $t^i_{S_1,P_1,A_1}$. When we have $t^i_{S_0,P_0,A_0} = t^i_{S_1,P_1,A_1}$, it means that $\forall x \in [0,R]$, $Pr\{t^i_{S_0,P_0,A_0} \le x\} = Pr\{t^i_{S_1,P_1,A_1} \le x\}$. Similarly, $t^i_{S_0,P_0,A_0} \le t^i_{S_1,P_1,A_1}$ is defined as $\forall x \in [0,R]$, $Pr\{t^i_{S_0,P_0,A_0} \le x\} \le Pr\{t^i_{S_1,P_1,A_1} \le x\}$. With the comparison rules between two random variables, we have the following theorem.

Theorem 4.1. *Let $\langle S^i_0, P^i_0, A^i_0\rangle$ and $\langle S^i_1, P^i_1, A^i_1\rangle$ be two solutions in the same cell of dynamic programming table. If $prob^i_{S_0,P_0,A_0} \le prob^i_{S_1,P_1,A_1}$ and $t^i_{S_0,P_0,A_0} \le t^i_{S_1,P_1,A_1}$, then we can remove the solution $(prob^i_{S_0,P_0,A_0}, t^i_{S_0,P_0,A_0})$.*

Proof. Assume the global optimal solution $\langle S_0, P_0, A_0\rangle$ contains the solution $\langle S^i_0, P^i_0, A^i_0\rangle$. Then, in the optimal solution $\langle S_0, P_0, A_0\rangle$ for the subgraph $v_1 \to v_i$, we replace S^i_0 by S^i_1, replace P^i_0 by P^i_1 and replace A^i_0 by A^i_1. Based on the above replacements, we can obtain a new solution, denoted as $\langle S_1, P_1, A_1\rangle$.

Since $\langle S^i_0, P^i_0, A^i_0\rangle$ and $\langle S^i_1, P^i_1, A^i_1\rangle$ have the same cost, the total costs of two solutions ($\langle S_0, P_0, A_0\rangle$ and $\langle S_1, P_1, A_1\rangle$) are the same. In addition, since the solution for the subgraph $v_{i+1} \to v_N$ are the same and we have the conditions (1) $prob^i_{S_0,P_0,A_0} \le prob^i_{S_1,P_1,A_1}$ and (2) $t^i_{S_0,P_0,A_0} \le t^i_{S_1,P_1,A_1}$, we can deduce that $Pr\{t^i_{S_1,P_1,A_1} \le R\} \ge Pr\{t^i_{S_0,P_0,A_0} \le R\} \ge GP$. Therefore, the solution $\langle S_1, P_1, A_1\rangle$ is no worse than $\langle S_0, P_0, A_0\rangle$ under the given timing constraints. In other words, the solution $\langle S_1, P_1, A_1\rangle$ is also optimal. □

Algorithm 2 The computation of $\boldsymbol{D_{i,j,k,e,l_1,l_2,m_1,m_2}}$ in probabilistic scenarios

```
Input: The computed dynamic programming cells, from node $v_1$ to $v_i$; the required iteration period $R$; the required guaranteed probability $GP$.
Output: The dynamic programming cell, $D_{i,j,k,e,l_1,l_2,m_1,m_2}$.
 1: $PosSolSet = \emptyset$;
    // Case 1: Add $v_i$ into the former partition.
 2: $\forall(prob^{i-1}_{S,P,A}, t^{i-1}_{S,P,A}) \in D_{i-1,j,k,e-ec_{c_j}(v_i,l_j),l_1,l_2,m_1,m_2}$;
 3:    $prob^i_{S,P,A} = prob^{i-1}_{S,P,A}$;
 4:    $t^i_{S,P,A} = t^{i-1}_{S,P,A} + et_{c_j}(v_i, l_j)$;
 5:    if $Pr\{t^i_{S,P,A} \le R\} \times prob^{i-1}_{S,P,A} \ge GP$:
 6:       Insert $(prob^i_{S,P,A}, t^i_{S,P,A})$ into $PosSolSet$
    // Case 2: Add $v_i$ into a new partition.
 7: $\forall c_{j^*} \in C$:
 8:    $\forall(prob^{i-1}_{S,P,A}, t^{i-1}_{S,P,A}) \in D_{i-1,j^*,k-1,e',l_1,l_2,m_1-1,m_2}$
       // where, $e' = e - ec_{cj(v_i,l_j)}$
 9:       $prob^i_{S,P,A} = Pr\{t^{i-1}_{S,P,A} \le R\} \times prob^{i-1}_{S,P,A}$
10:       $t^i_{S,P,A} = \delta + et_{c_j}(v_i, l_j)$;
11:       if $prob^i_{S,P,A} \ge GP$ and $Pr\{t^i_{S,P,A} \le R\} \ge GP$:
12:          $Insert$ $(prob^i_{S,P,A}, t^i_{S,P,A})$ $into\ PosSolSet$;
    // Finally, select the optimal (superior) solution(s).
13: if $PosSolSet = \emptyset$:
14:    return $(\infty, \infty)$;
15: else
16:    Remove redundant solutions in $PosSolSet$;
17:    return $PosSolSet$;
```

Now, we are ready to introduce the algorithm to obtain the optimal solutions in probabilistic scenarios. The algorithm is presented in Algorithm 2. The basic idea of this algorithm is similar with that in Algorithm 1. There are two cases when considering node v_i.

In **Case** 1, we add v_i into the partition containing v_{i-1}. Since each cell in dynamic programming table contains a set of superior sub-solutions, we need to traverse the set as shown in line 2. Since node v_i is added into the former partition, the product of probabilities of all complete partitions is not changed (in line 3). And the time of incomplete partition is enlarged, which is computed in line 4. In line 5, we check whether the current assignment can satisfy the timing constraint R with guaranteed probability GP (we assume the incomplete stage will be not expanded). And in line 6, we add the solution into the possible solution set $PosSolSet$.

In **Case** 2, a new complete partition and a new incomplete partition is formed. We update the product of probabilities of all complete partitions in line 9. And the latency of new partition is initialized as the sum of synchronization overhead and execution time of node v_i (line 10). We add feasible solutions into $PosSolSet$ in line 12. Finally, in lines 13 to 17, we prune the $PosSolSet$ and return the superior solutions.

The next theorem shows that each step in our algorithms will obtain the local optimal solutions, and in the end of the algorithm we will obtain the global optimal solution.

Theorem 4.2. *In Algorithm 1, if e is the minimum energy consumption that $D_{i,j,k,e,l_1,l_2,m_1,m_2} \ne \emptyset$, then it is impossible to obtain feasible solutions, such that the energy of the subgraph from v_1 to v_i is less than e.*

The above theorem can be proved by induction. Let e_{min} be the minimum energy that $D_{N,j,k,e_{min},l_1,l_2,m_1,m_2} \neq \emptyset$ where N is the last node in task graph. Based on Theorem 4.2, we know that for all selections, partitions, and assignments, the minimum energy of all feasible solutions is e_{min}.

The time complexity of the algorithm is $O(N \cdot C \cdot P \cdot E \cdot L^C \cdot CR^C \cdot K^3)$, where N, C, P, L, CR represent the maximum number of nodes, clusters, partitions, V/F levels in clusters, cores in clusters, respectively. Notation E is the number of possible values of energy consumption and notation K is the maximum number of $(prob^i_{S,P,A}, t^i_{S,P,A})$ pairs in each cell of dynamic programming table ($K = 1$ in fix-time scenarios). Our algorithm is practical for existing platforms with 2 or 3 clusters (MediaTek; SMP 2011; Odroid-XU3). Furthermore, it is scalable and capable for the platform integrated with many cores, since the number of clusters is drastically less than core numbers.

5. Experimental Results

In this section, we conduct comprehensive experiments to show that our presented algorithms can effectively obtain the optimal solutions in probabilistic scenarios, achieving significant reductions in total energy consumption. In addition, the elapsed time of our proposed algorithm is much less than that of other existing techniques. Results reflect the efficiency and practicability of our algorithms.

5.1 Experimental Setup

All experiments are conducted on a real mobile platform, Odroid-XU3 (Odroid-XU3). It implements ARM big.LITTLE technique with two clusters of four ARM Cortex-A15 cores and four ARM Cortex-A7 cores. The platform runs popular Ubuntu 14.04 LTS operating system. We can change the frequency of each cluster by editing virtual files in the operating system, and the voltage is scaled automatically by hardware corresponding to the frequency. In our experiments, we use 6 available V/F levels for each cluster, as shown in Table 1.

We base our experiments on 8 benchmarks in StreamIT (Thies et al. 2002), including key generation for DES encryption algorithm (denoted by `deskey`, having 12 nodes), DES encryption algorithm (denoted by `des`, having 10 nodes), serpent encryption algorithm (denoted by `serpent`, having 13 nodes), JPEG encoder (denoted by `jpeg`, having 17 nodes), blocked matrix multiply (denoted by `matmul`, having 13 nodes), vocoder for bitrate reduction in speech (denoted by `vocoder`, having 14 nodes), partial MP3 decoder (denoted by `mp3`, having 14 nodes), and MPEG2 encoder (denoted by `mpeg2`, having 12 nodes). We trace the execution of tasks by running benchmarks on Odroid-XU3 platform. And we build historic table to obtain the probabilistic execution times; meanwhile, we obtain the energy consumption of each task.

Based on these information, we compare our optimal algorithm with 3 existing approaches: (1) the ILP formulation using worst-case execution time (Kuang et al. 2005) (called "Fix-ILP"). (2) the two-phase approach presented in (Niu et al. 2014) (called "Two-Phase"), which separately considers task assignment and V/F selection in probabilistic scenarios. (3) the genetic algorithm based on the approach presented in (Wang et al. 2011) (called "GeneS").

In the experiments, we compare different approaches in two metrics: the total energy consumption of resultant systems and the running time of different algorithms. Experiments will show that our algorithms outperform the existing approaches significantly. In the meanwhile, it only takes seconds to obtain the optimal solution, while others takes minutes to obtain the near-optimal solutions. Benefit from the efficiency of our proposed algorithms, we can effectively explore the design space. We also compare the obtained results with that generated by random approach. The exploration on design spaces provide opportunities for designers to investigate the tradeoff between cost and performance.

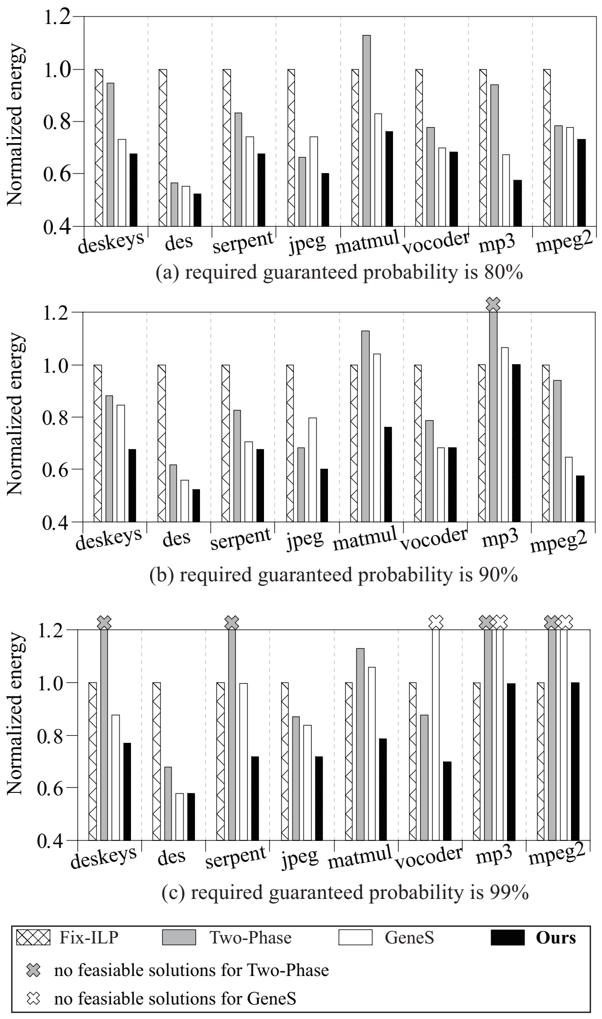

Figure 4. Comparisons on total energy consumption under different Required iteration periods (R) and Guaranteed Probabilities (GP). Required iteration periods are set as the same with Table 3. Required guaranteed probabilities are: (a) GP = 80%; (b) GP = 90%; (c) GP = 99%.

5.2 Energy Consumption and Running Times

Figure 4 reports the comparisons in energy consumption with different guaranteed probabilities. The x-axis represents the benchmarks and y-axis represents the normalized energy consumption. In the results, the baseline is set as the results of Fix-ILP approach.

As shown in the results, our approach continuously achieves 21.73%, 31.33% and 34.64% reductions compared with Fix-ILP, when the required guaranteed probabilities are 99%, 90% and 80%, respectively. Compared with Two-Phase and GeneS approaches, the average energy consumption savings are 23.93% and 13.91%, respectively. In addition, there are many cases that Two-Phase and GeneS approaches cannot generate feasible solutions.

From the results, there are several observations.

- In many cases in Figure 4(c), Two-Phase and GeneS approaches may not find any feasible solutions, but ours can. The reason is that these approaches cannot explore design spaces thoroughly.

Table 4. Running times (secs).

Bench.	Fix-ILP	Two-Phase	GeneS	**Ours**
`deskey`	22320	47.73	359.61	**10.21**
`des`	1594	53.61	495.9	**7.35**
`serpent`	16920	181.5	1490.1	**15.36**
`jpeg`	320	3,28	155.38	**1.11**
`matmul`	408	52.17	933.76	**14.34**
`vocoder`	2	1.35	161.63	**0.35**
`mp3`	77	18.90	392.32	**6.30**
`mpeg2`	23	25.03	306.53	**2.95**

- For those cases that other approaches can find feasible solutions, our algorithms can achieve significant reductions in energy consumption. Specifically, when $GP = 90\%$, reductions are 31.33%, 26.34% and 16.37% on average compared with Fix-ILP, Two-Phase and GeneS, respectively.
- Results of Fix-ILP are inferior to ours, because it does not consider the probabilistic features in execution time. It emphasizes the importance of minimizing energy consumption in the probabilistic scenarios.
- Results of Two-Phase approach are inferior to GeneS and Ours. Because separate optimization on partitions, assignments, and selections may lead to local optimal solutions. In other words, three decision functions are highly coupled with each other and they should be taken into consideration simultaneously.

These results clearly demonstrate the effectiveness of our proposed algorithms to obtain the optimal solutions.

Table 4 records the running times of different approaches. As we can see in this table, for all benchmarks, our algorithm can obtain the optimal solution in seconds. Whereas, GeneS cannot obtain a solution within one minute. For Fix-ILP, as the increase of tasks in benchmarks, its running time increases exponentially. In particular, for `deskey` with 24 nodes, it requires 6 hours to get feasible solutions. Results show our proposed algorithms are efficient and practical.

5.3 Design Space Exploration

Our algorithms can efficiently and optimally explore design spaces; i.e., we can obtain the Pareto fronts. The Pareto front is a series of optimal solutions (or called design points) obtained by specifying the required iteration periods under a given probability. For instance, we obtain the Pareto fronts of `jpeg` by specifying iteration periods from 37us to 137us in step of 1us. For each guaranteed probability, we employ the presented algorithms to obtain the optimal design points.

Results of `jpeg` are reported in Figure 5. In these figures, the x-axis represents different required iteration periods (us), and y-axis represents the total energy consumption (uJ). The dots in these figures represent the results obtained by random approach.

Results in Figure 5(a)-5(c) illustrate the Pareto fronts under each guaranteed probability and the design points generated by random approach. In random approach, we generate $|V|^2$ ($|V|$ represents the number of tasks) solutions for each iteration period, and we keep the feasible solutions, represented by dots. Results clearly show that random approach cannot generate optimal design points due to the large design space. Another observation is that random approach cannot find feasible solutions in tight timing constraints, but ours can. Because our algorithm can fully explore the design space. The above results show that our algorithms can effectively obtain Pareto fronts for each required probability.

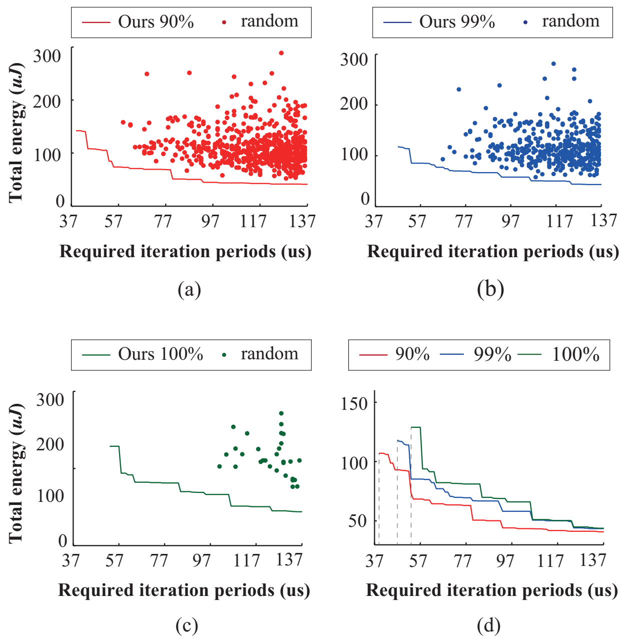

Figure 5. Design space exploration for `jpeg` under 3 different Guaranteed Probabilities: (a) GP=90% (soft real-time systems); (b) GP=99% (soft real-time systems); (c) GP=100% (hard real-time systems); (d) Comparisons on Pareto fronts under different guaranteed probabilities

Figure 5(d) illustrates Pareto fronts with different probabilities. It demonstrates the advantages of our algorithms in two aspects. (1) Compared with worst-case scenarios, systems constructed by our algorithms can always achieve less energy consumption. (2) For tight timing constraints (38us to 53us), there exist no feasible solutions in worst-case scenarios. In probabilistic scenarios, however, we can obtain solutions with a high guaranteed probability. Results show that our algorithms can achieve significant reductions in energy consumption and provide more design choices.

6. Related Work

With the growing demand for high-performance and low-cost design, embedded devices move toward heterogeneous Multiprocessor System-on-Chip (MPSoC) platforms (Xie and Hung 2006; Wang et al. 2011; Yang et al. 2016a; Donyanavard et al. 2016; Prakash et al. 2015). On such platforms, the applications can be executed in a pipelined fashion, such as streaming applications (Javaid et al. 2014; Jiang et al. 2016) and convolutional neural network applications (Chen et al. 2016; LeCun et al. 2010). This promotes the rise of a specialized MPSoC, called pipelined MPSoC (Carta et al. 2007; Javaid et al. 2010). Pipelined MPSoCs equip a single or multiple processors to host the system and multiple heterogeneous functional units (cores). These functional units have different cost in terms of performance.

On pipelined MPSoCs, the execution time of tasks is usually not fixed but has a probability to be finished in a certain time, due to the effects of conditional instructions, caches, fluctuant I/O loads, system interruptions, and thread synchronization (Tongsima et al. 2000; Hung et al. 2005; Xie and Hung 2006; Hua et al. 2003; Hu et al. 2013; Catthoor et al. 2013). Therefore, the variation of execution time of each task in applications has to be taken into consideration. In the existing research efforts, execution times are considered in two scenarios: *worst-case scenario*, the worst-case execution time is used; *probabilistic scenario*, the execution time is modeled as random variables.

Most research efforts employ worst-case execution time to construct systems. (Chao and Sha 1997; Karkowski and Corporaal 1997; Ito et al. 1998; Bakshi and Gajski 1999; Kuang et al. 2005; Shao et al. 2005; Gordon et al. 2006; Cucinotta and Palopoli 2010; Javaid et al. 2010, 2014; Prakash et al. 2015; Yang et al. 2016b; Jiang et al. 2017; Yang et al. 2017). Bakshi and Gajski (1999) develop heuristic algorithms to partition a system to minimize area under performance constraint. Their results, however, cannot obtain the optimal solution. Ito et al. (1998) propose an Integer Linear Programming (ILP) model to assign nodes to heterogeneous functional units. Kuang et al. (2005) present ILP models to partition and schedule streaming applications. ILP formulations take too long to obtain a solution. Shao et al. (2005) present efficient techniques for assignments problem. Gordon et al. (2006) conduct assignment to achieve pipeline parallelism in homogenous architectures. Javaid et al. (2010, 2014) target on the assignment of the partitioned applications on pipelined heterogeneous MPSoC. Prakash et al. (2015) present static partition strategy on heterogeneous mobile platforms. None of the above techniques consider the variations of execution time. The worst-case execution time is widely used in fix-time scenario to meet the timing constraints, which is overly pessimistic in the design of soft real-time systems.

Recently, more studies focus on optimizing systems in probabilistic scenarios (Zhu et al. 2008; Qiu and Sha 2009; Cucu-Grosjean et al. 2012; Niu et al. 2014; Jalle et al. 2014; Sha et al. 2015; Abeni et al. 2016; Jiang et al. 2016). Qiu and Sha (2009) consider the heterogeneous assignment problem in the probabilistic scenario. Niu et al. (2014) devise two-phase algorithm to conduct assignment and voltage selection. Sha et al. (2015) present algorithms to reduce energy consumption by using dynamic voltage scaling technique. Abeni et al. (2016) analyze the pipelines with probabilistic deadlines. Jiang et al. (2016) present techniques to conduct functional-unit assignment. In all the above researches, the execution time is modeled based on historical data. In addition, these work partially considers the assignment or voltage selection. To the best of authors' knowledge, this is the first paper that figures out the feature on the execution times of tasks; i.e., the varying execution times of a task are gathered in clusters. Based on this feature, this paper models the probabilistic execution times of tasks as discrete random variables.

Optimizations in probabilistic scenarios are much harder than that in worst-case scenario due to the large design space (Qiu and Sha 2009; Jiang et al. 2016). In addition, the optimization of pipelined heterogeneous MPSoCs is much more challenging since the determinations on partitions, functional unit assignments, and voltage selections are coupled to each other. In order to achieve the global optimal solution, these determinations should be made simultaneously. As stated, many existing research efforts partially consider these decision functions or separately optimize these decisions. In consequence, these problems can be regarded as a special case of the problem solved in this paper.

7. Conclusion

Pipelined MPSoCs provide high performance for applications by overlapping executions in different iterations. In the design of pipelined heterogenous MPSoCs, there are three basic issues that need to be studied: in what degree the system is pipelined, which related to the partition problem; on what type of functional unit to execute each pipeline stage, which related to the assignment problem; and using what voltage/frequency level for each functional unit, which related to the selection problem. Since the above three problems are not independent but coupled, it requires designers to tackle these problems simultaneously to obtain optimal solutions.

Due to the increasing complexity of applications, designers face a bundle of problems in synthesizing pipelined MPSoCs. One of the most challenging problems is the uncertainties on the execution times, which makes the deterministic techniques inapplicable. In this work, we presented algorithms to optimally construct pipelined MPSoCs. We first found the features of probabilistic execution times on real-world mobile platforms. Based on our observations, we modeled execution times of tasks as random variables. Then, we proposed efficient algorithms to find the optimal solutions. Experiments show that our algorithms can significantly reduce cost, compared with other approaches. In addition, we can obtain solutions in seconds, while other approaches need minutes or even hours to obtain inferior solutions. Furthermore, our algorithms can efficiently explore the design spaces to obtain the Pareto fronts, which cannot be given by other existing techniques.

In the future, we plan to incorporate the probabilistic model in scheduling problems. We also plan to investigate how to guarantee the system performance with high probability that can tolerant faults in the execution, which may significantly save energy while maintaining the system reliability.

Acknowledgments

This work is partially supported by the National 863 Program 2015AA015304, NSFC 61472052, Chongqing High-Tech Research Program cstc2014yykfB40007

References

L. Abeni et al. Stochastic analysis of buffer–less pipelines of real–time tasks. In *ACM Symposium On Applied Computing (SAC), 2016*, pages 1–8. ACM, 2016.

S. Bakshi and D. D. Gajski. Partitioning and pipelining for performance-constrained hardware/software systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(4):419–432, 1999.

S. Carta, A. Alimonda, A. Pisano, A. Acquaviva, and L. Benini. A control theoretic approach to energy-efficient pipelined computation in mpsocs. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(4):27, 2007.

F. Catthoor, S. Wuytack, G. de Greef, F. Banica, L. Nachtergaele, and A. Vandecappelle. *Custom memory management methodology: Exploration of memory organisation for embedded multimedia system design*. Springer Science & Business Media, 2013.

L.-F. Chao and E. H.-M. Sha. Scheduling data-flow graphs via retiming and unfolding. *IEEE Transactions on Parallel and Distributed Systems*, 8 (12):1259–1267, 1997.

Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 367–379. IEEE, 2016.

T. Cucinotta and L. Palopoli. Qos control for pipelines of tasks using multiple resources. *IEEE Transactions on Computers*, 59(3):416–430, 2010.

L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 91–101. IEEE, 2012.

B. Donyanavard, T. Mück, S. Sarma, and N. Dutt. Sparta: runtime task allocation for energy efficient heterogeneous many-cores. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 27. ACM, 2016.

M. I. Gordon, W. Thies, and S. Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. *ACM SIGOPS Operating Systems Review*, 40(5):151–162, 2006.

J. Hu, Q. Zhuge, C. J. Xue, W.-C. Tseng, and E. H.-M. Sha. Software enabled wear-leveling for hybrid pcm main memory on embedded systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 599–602. IEEE, 2013.

S. Hua, G. Qu, and S. S. Bhattacharyya. Energy reduction techniques for multimedia applications with tolerance to deadline misses. In *Proceedings of the 40th annual Design Automation Conference*, pages 131–136. ACM, 2003.

W.-L. Hung, Y. Xie, N. ViJ'aykrishnan, M. Kandemir, and M. J. Irwin. Thermal-aware task allocation and scheduling for embedded systems. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 898–899. IEEE, 2005.

K. Ito, L. E. Lucke, and K. K. Parhi. Ilp-based cost-optimal dsp synthesis with module selection and data format conversion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):582–594, 1998.

J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Bus designs for time-probabilistic multicore processors. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 50. European Design and Automation Association, 2014.

H. Javaid, X. He, A. Ignjatovic, and S. Parameswaran. Optimal synthesis of latency and throughput constrained pipelined mpsocs targeting streaming applications. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 75–84. IEEE, 2010.

H. Javaid, A. Ignjatovic, and S. Parameswaran. Performance estimation of pipelined multiprocessor system-on-chips (mpsocs). *IEEE Transactions on Parallel and Distributed Systems*, 25(8):2159–2168, 2014.

W. Jiang, E. H.-M. Sha, Q. Zhuge, and X. Chen. Optimal functional-unit assignment and buffer placement for probabilistic pipelines. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2016 International Conference on*, pages 1–10. IEEE, 2016.

W. Jiang, E. H.-M. Sha, X. Chen, L. Yang, L. Zhou, and Q. Zhuge. Optimal functional-unit assignment for heterogeneous systems under timing constraint. *IEEE Transactions on Parallel and Distributed Systems*, 2017.

I. Karkowski and H. Corporaal. Design of heterogenous multi-processor embedded systems: applying functional pipelining. In *Parallel Architectures and Compilation Techniques., 1997. Proceedings., 1997 International Conference on*, pages 156–165. IEEE, 1997.

S.-R. Kuang, C.-Y. Chen, and R.-Z. Liao. Partitioning and pipelined scheduling of embedded system using integer linear programming. In *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on*, volume 2, pages 37–41. IEEE, 2005.

Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.

MediaTek. Mediatek helio x20. `http://mediatek-helio.com/x20/`.

J. Niu, C. Liu, Y. Gao, and M. Qiu. Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(8): 2043–2052, 2014.

Odroid-XU3. Odroid-xu3. `http://goo.gl/Nn6z3O`.

A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 208–215. IEEE, 2015.

M. Qiu and E. H.-M. Sha. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(2):25, 2009.

M. Qiu, C. Xue, Q. Zhuge, Z. Shao, M. Liu, and E. H.-M. Sha. Voltage assignment and loop scheduling for energy minimization while satisfying timing constraint with guaranteed probability. In *Application-specific Systems, Architectures and Processors, 2006. ASAP'06. International Conference on*, pages 178–181. IEEE, 2006.

A. Salman, I. Ahmad, and S. Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8): 363–371, 2002.

E. H.-M. Sha, W. Jiang, Q. Zhuge, L. Yang, and X. Chen. On the design of high-performance and energy-efficient probabilistic self-timed systems. In *High Performance Computing and Communications (HPCC), 2015*, pages 260–265. IEEE, 2015.

Z. Shao, Q. Zhuge, C. Xue, and E.-M. Sha. Efficient assignment and scheduling for heterogeneous dsp systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):516–525, 2005.

N. V. SMP. A multi-core cpu architecture for low power and high performance. Technical report, NVidia,, 2011.

W. Thies, M. Karczmarek, and S. Amarasinghe. Streamit: A language for streaming applications. In *International Conference on Compiler Construction*, pages 179–196. Springer, 2002.

S. Tongsima, E.-M. Sha, C. Chantrapornchai, D. R. Surma, and N. L. Passos. Probabilistic loop scheduling for applications with uncertain execution time. *IEEE transactions on computers*, 49(1):65–80, 2000.

Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, and E. H.-M. Sha. Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 16(2):14, 2011.

Y. Xie and W.-L. Hung. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design. *The Journal of VLSI Signal Processing*, 45(3):177–189, 2006.

L. Yang, W. Liu, W. Jiang, M. Li, P. Chen, and E. Sha. Fotonoc: A folded torus-like network-on-chip based many-core systems-on-chip in the dark silicon era. *IEEE Transactions on Parallel and Distributed Systems*, 2016a.

L. Yang, W. Liu, W. Jiang, M. Li, J. Yi, and E. H.-M. Sha. Application mapping and scheduling for network-on-chip-based multiprocessor system-on-chip with fine-grain communication optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10): 3027–3040, 2016b.

L. Yang, W. Liu, W. Jiang, C. Chen, M. Li, P. Chen, and H. Edwin. Hardware-software collaboration for dark silicon heterogeneous many-core systems. *Future Generation Computer Systems*, 68:234–247, 2017.

D. Zhu, H. Aydin, and J.-J. Chen. Optimistic reliability aware energy management for real-time tasks with probabilistic execution times. In *Real-Time Systems Symposium, 2008*, pages 313–322. IEEE, 2008.