Evaluation Methodology for Data Communication-Aware Application Partitioning

Imran Ashraf, S. Arash Ostadzadeh, Roel Meeuws, and Koen Bertels

Computer Science and Engineering Group,
Department of Software and Computer Technology,
Delft University of Technology, Delft, The Netherlands
{I.Ashraf,S.A.Ostadzadeh,R.J.Meeuws,K.L.M.Bertels}@TUDelft.nl

Abstract. Heterogeneous multicore architectures appear to be a promising solution to the ever-increasing computational demands of applications in embedded as well as high-performance computing domains. Nevertheless, inefficient application partitioning for such systems may result in a considerable reduction in the anticipated performance improvement. Consequently, proper mechanisms are required to evaluate quality of different partitions in an early phase of application mapping. This evaluation is especially important in the case of reconfigurable accelerators, as the synthesis phase of their hardware blocks is quite time-consuming. Hence, an early evaluation of application partitioning mechanisms for such architectures can reduce time-to-market.

In this paper, we propose a data communication-aware methodology for evaluating the quality of application partitions as well as partitioning algorithms. We also present an open source tool which implements the proposed methodology. Moreover, we evaluate several heuristic algorithms to further substantiate the applicability of the proposed methodology and the utilization of the developed tool. The applications are used as test inputs for the sake of comparisons in terms of relative and absolute quality measurements of the partitioning solutions.

Keywords: Application partitioning, Performance evaluation, Heuristic algorithms, Data communication analysis.

1 Introduction

Heterogeneous multicore architectures are gaining popularity in embedded devices as well as high-performance computers [1]. This new trend poses specific challenges regarding the programmability of these architectures. One such challenge is application partitioning, in which an application is divided into smaller parts to be mapped onto various Processing Elements (PEs). This is a critical task because inefficient partitioning may reduce the anticipated performance improvement. It has also been discussed that the communication among various parts of an application entails a major design challenge [2,3].

The criteria that drive application partitioning include, among others, the nature of computations, execution times on PEs, memory requirements, area

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 739–748, 2014.

avaialble on PEs, etc. Due to the huge size of the search space, finding an optimal *partition* is an NP-hard problem [4]. Therefore, heuristic algorithms are commonly utilized to find solutions in short time. Nevertheless, the *partition* found by a heuristic may or may not be close to the optimal solution. Hence, it is very important to be able to evaluate the quality of the solution(s) found by a heuristic method and to make a robust comparison with the solutions found by other existing or future partitioning algorithms.

Due to a large variety of architectures and the lack of proper benchmarks, it is hard to reproduce experimental results, for fair comparison, on the target platforms [5]. Moreover, the complexity furthers in case of reconfigurable architectures as the application development process involves building and synthesizing hardware blocks. Hence, a proper methodology is required to quickly evaluate the quality of the partitioning algorithms.

An interesting observation is that only a small number of functions primarily contribute to the overall application execution time. Similarly, a limited number of functions are responsible for the main inter-task data communication. By considering these two observations, we can potentially reduce the huge design space exploration effort. Thus, the comparison with optimal partitions found by an exhaustive search becomes feasible, in spite of the general intractability of the application partitioning problem.

In this work, we propose a methodology for evaluating partitions found by various application partitioning algorithms. Furthermore, we compare the outcome of several partitioning algorithms based on the quality of the found solutions. We define cost metrics based on the execution contribution of the functions and the data communication. The execution contribution and data communication information are obtained by profiling the application [6][7]. Our main contributions are summarized as follows:

- formulation of the proposed data communication-aware application partitioning evaluation methodology;
- design and implementation of an open source tool that implements this methodology;
- evaluation of a multi-objective partitioning algorithm as a case study for synthetic applications and real benchmarks.

The rest of the paper is structured as follows. Section 2 overviews the related research work. Section 3 presents the proposed evaluation methodology. Section 4 describes the modular design of the *partition* evaluation tool. Subsequently, Section 5 provides a detailed case study describing the evaluation of a multi-objective partitioning algorithm to illustrate the methodology. Section 6 details the experimental results. Finally, Section 7 provides the conclusions and future research directions.

2 Related Work

Efficient application partitioning for multicore platforms has been an active field of research in the last decade. The problem has been approached in diverse

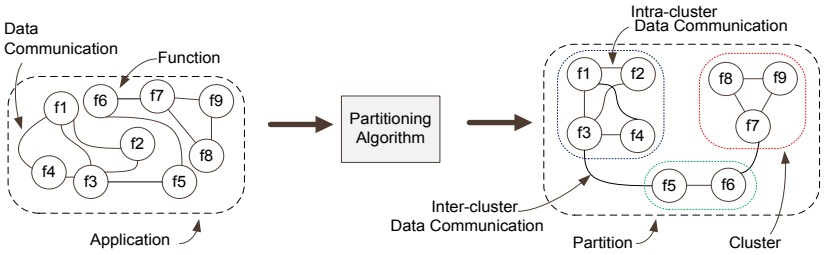

Fig. 1. A partitioning algorithm takes an application as input and produces a *partition* at the output. The depicted *partition* is composed of three clusters.

ways and at various granularity levels, ranging from fine-grained (basic blocks or loops) to coarse-grained (functions), heuristic and evolutionary approaches, as reviewed in [7]. The work presented in [6] utilizes a data communication graph, annotated with area estimates to do partitioning in a semi-automatic way. A generalized framework for automatic code partitioning is discussed in [8]. Some works regarding code partitioning for platforms containing reconfigurable accelerators are reviewed in [7].

An empirical evaluation is provided in [9], where graphs are compared using Chaco framework [10]. Communication costs among vertices in these graphs are based on cut edges. The quality of a *partition* is found by the application execution time on the target platform. However, finding the quality of *partitions* by executing the application on reconfigurable platforms, if possible in all cases, is a tedious and time-consuming task, as it involves synthesizing hardware blocks for reconfigurable units. Our methodology is able to perform an early evaluation without the need to run on the target platform.

Another evaluation is provided in [5], which presents a comparison of four partitioning approaches. The authors associate cost metrics with *partitions* and provide a comparison of these approaches. However, their focus is on the hardware/software partitioning, which is a subset of the generalized partitioning problem. Our work provides a generic evaluation methodology for any number of parts in a partition.

3 Evaluation Methodology

Figure 1 depicts the application partitioning process and the terminology used in the proposed evaluation methodology, which is a 4-step process described below.
Step 1: Formulation of the cost function. Partitions can be evaluated by calculating their costs using a cost function. A cost function takes a *partition* as input and assesses its quality. Various factors can contribute to the quality of a *partition*, for instance, how well the clusters are balanced, inter-cluster data communication, etc. In this step, the factors of interest are determined to formulate the cost function.
Step 2: Implementation of the partitioning algorithm. The partitioning algorithm to be evaluated is implemented in this step. The algorithm takes the application as input and outputs a *partition* of the application.

Step 3: Specification of the input set. In this step, the applications that are input to the partitioning algorithm are specified. These applications are represented as graphs, where the values of graph vertices represent the characteristics of applications, for instance, execution contribution, memory requirements, area estimates, etc. The edges in these graphs represent the data communication between functions. The specification of an input application boils down to selecting numbers for vertices and edges, which can be obtained randomly or by profiling real applications.
Step 4: Evaluation and comparison. In this step, the cost of the *partition* found by the *partitioning algorithm* is evaluated by the cost function. The evaluated cost can be used to perform comparisons and to rank the *partition* based upon its quality. This ranking can be:

Absolute — where the comparison is performed with the optimal *partition* found by an exact solution or an exhaustive search (only for applications with small number of functions).

Relative — when optimal *partitions* cannot be found, thus the cost is compared in relation to the costs of *partitions* found by other algorithms. In the next section, we present the details of the tool that implements the proposed methodology.

4 PET Implementation

We developed the Partition Evaluation Tool (PET[1]), to evaluate the quality of partitions and compare various partitioning algorithms. This tool is implemented in C++ in a flexible way to allow the partitioning strategies to be evaluated easily. We define the relevant concepts used in the PET discussion as follows.

Application class models the concept of the application which needs to be partitioned by the partitioning algorithm. An application has two important members, namely functions and edges.
Function represents the subroutine in an application, which performs a certain task of the application. Each function has an execution contribution depending upon the task assigned to it.
Edge is a directed link denoting the communication between a pair of functions in an application. The amount of data that is communicated is represented by the weight of the edge.
Cluster represents the collection of functions which can be mapped onto a single core/PE of the target platform. The number of functions in a cluster is controlled by the capacity of that cluster. As an example, in reconfigurable systems, this capacity may represent the maximum number of slices reserved for a single PE. When a cluster is full, its status is changed from *UnFinished* to *Finished*.
Partition refers to the union of clusters, where each cluster contains various functions. A viable partition holds the semantics of the application with the combination of functions in clusters that can be mapped onto various cores/PEs

[1] Sources available at `http://imranashraf.github.com/PET`

in a multicore platform. The cost of a *partition* can be evaluated by a call to the *Cost()* method of this class.
PartitioningAlgorithm is an abstract class that can be used to implement the algorithm(s) used to perform partitioning. This can be achieved by implementing the *Apply()* virtual function of the *PartitioningAlgorithm* class for some algorithm of interest.

5 Case Study

In this section, we present the evaluation of a multi-objective task clustering algorithm [7], to illustrate our partition evaluation methodology and the utilization of PET in this regard. This algorithm initiates task clustering at the function-level based on dynamic profiling information. This includes the data communication among functions and the information about their execution time. The overall goal of the heuristic algorithm is to get a well-balanced cluster containing tightly inter-connected functions. The steps of the evaluation methodology are described below:
Step 1: Formulation of the Cost Function. We assume an application containing n functions to be partitioned into k clusters. These functions can be specified as vertices $v_i, 1 \leq i \leq n$ in a graph. The execution cost EC_{v_i} of a vertex v_i, is the cost of executing this function on a PE when it is mapped as a part of a cluster. The execution cost of a cluster C_i is defined as: $EC_{C_i} = \sum_{v_j \in C_i} EC_{v_j}, 1 \leq i \leq k, 1 \leq j \leq n$.

Vertices v_i and v_j have an edge e_{ij} connecting them, with the weight w_{ij} representing the amount of the data that is communicated. The set of all the edges in a graph makes an edge set E of the graph. The set of edges which cross the boundary of a cluster C_i form the set of external edges $Eext_{C_i}$ of this cluster with respect to the rest of the partition. This edge set represents the communication cost of the cluster C_i with respect to the other clusters. Furthermore, the set of edges internal to a cluster C_i form the internal edge set $Eint_{C_i}$, as it represents the communication among functions inside C_i. For this case study, we consider the following two metrics to formulate the total cost of a *partition*:

1. **Balancing Penalty (BP)** accounts for the load balancing among clusters in a partition. It basically depends upon the distance between the execution cost EC_{C_i} of cluster C_i and the average execution cost of all the clusters in a partition $BP_{C_i} = \left| \frac{\sum_{j=1}^{k} EC_{C_j}}{k} - EC_{C_i} \right|, 1 \leq i \leq k$. Balancing penalty BP_P of a partition P is defined as $BP_P = \sum_{i=1}^{k} BP_{C_i}$.
2. **Communication Cost (CC)** is the cost associated with external communication of a cluster with respect to the other clusters in the partition. Communication cost CC_{C_i} of a cluster C_i is $CC_{C_i} = \sum_{e_{ij} \in Eext_{C_i}} w_{ij}$. The communication cost CC_P of a partition P is defined as $CC_P = \sum_{i=1}^{k} CC_{C_i}$.

The total cost TC_P of a partition P is then defined as:

$$TC = \alpha BP_P + \beta CC_P;\ 0 \leq \alpha \leq 1, 0 \leq \beta \leq 1 \ and\ \alpha + \beta = 1, \tag{1}$$

where α and β are relative weights associated with BP and CC metrics, respectively. These weights can be selected by the designer based on the platform at hand, to stress one metric relative to the other. For instance, if workload balancing is important then, one can select a higher value of α than β. On the other hand,when communication is expensive, β can get a higher value than α.

It is worth mentioning here that this is just one way of specifying the cost function as the weighted sum of the objectives of interest. Our approach is similar to the one discussed in [11], where the difference of values of each objective is explained in detail. On the same lines, we have assigned the values to each objective as their percentage contribution in the total execution and communication cost for BP and CC, respectively, instead of using their actual values.

Step 2: Implementation of the Partitioning Algorithm. We implemented the clustering algorithm as a derived class of the abstract *PartitioningAlgorithm* class of PET. The *Apply()* function implements the functionality of this heuristic algorithm on an input application. The result is a partition of the input application containing the clusters with various functions.

Step 3: Specification of the Input Set. We generated a number of synthetic application graphs to be used as test inputs. Due to space limitation, we will only discuss the real applications. We used applications from Standford Parallel Applications for Shared Memory Architectures (SPLASH-2) Benchmark Suite [12]. We used the MAIP and QUAD tools from Q^2 profiling toolset [6] to extract the percentage contribution of functions and to get the communication graphs. It should be noted that getting the profiling data by using any other profilers will not change the methodology or the evaluation process.

Step 4: Evaluation and Comparison. We performed the absolute ranking by finding the optimal *partition* determined by the exhaustive search. We compared the cost of *partition* found by the heuristic algorithm against the best possible partition, using *bruteforce()* class to generate all the possible partitions for the generated applications. The cost of each *partition* is evaluated by the *Cost()* method of the *Partition* class. Apart from the absolute ranking, we also performed relative ranking of the *partitioning algorithm* against the best solutions found by simulated annealing, evolutionary and tabu search. The detailed results of these evaluations and comparisons are provided in the following section.

6 Experimental Results

In this section, we provide the results of the experiments performed to evaluate the cost of heuristic algorithm and the cost of *partitions* found by the exhaustive search. We performed these experiments on a 2.66 *GHz Intel(R) Core(TM)2 Quad* CPU, with 4 *GB* RAM. We used the value of 0.5 for α and β to give equal weights to both cost metrics. Values given to these relative weights should be based on the target platform and care must be taken while revising them.

In order to present the evaluation for real benchmark applications, we selected three applications from SPLASH-2 benchmark suite, namely *Barnes*, *Fmm* and *Raytrace*. We also selected *Clustalw* [13] sequence alignment application and *KLT* [14] feature tracking application for the evaluation. Figure 2 presents the comparison of the cost of the partition found by Heuristic Search (HS) with the costs of the solutions found by Simulated Annealing (SA), Tabu Search (TS) and Evolutionary Search (ES). It can be seen that HS performs close to other algorithms. For *Barnes* and *Fmm*, which are relatively smaller applications, SA performs better, but for the remaining three bigger applications, TS outperforms all the other algorithms.

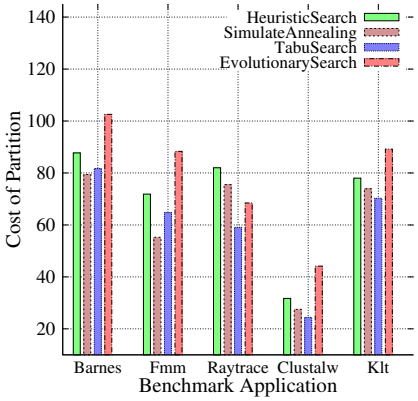

Fig. 2. Comparison of cost of partition found by HS against the cost of partition found by SA, TS and ES for the real applications

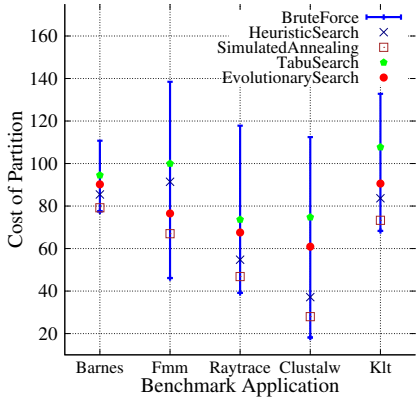

Fig. 3. Absolute ranking of the cost of the partition found by HS, SA, TS and ES from the real applications

An important point worth mentioning is that HS performs comparable to SA, TS and ES in terms of cost.However, HS takes very less time to find the solution compared to other algorithms. To give an idea, for the *KLT* application, HS took $815\mu s$ to find the solution, whereas, TS took $18m$.

In the above experiments, we performed evaluations relative to other algorithms. Absolute ranking can also be performed by comparing the costs against the cost of the optimal solution found by Brute Force (BF). Performing this exhaustive search to find the optimal *partition* of four clusters will require about 8 years on the computer used in these experiments for an application with 25 functions. The considered applications contain 38 to 103 functions, hence, performing the exhaustive search for the optimal *partition* is not practical. Thus, we limited the number of functions to a threshold value of 18, picking the top functions according to the execution time and data communication contribution. This feature is also supported by the tool, where we can specify this threshold value in terms of the number of functions for execution contribution and communication

Table 1. Specifications of the Real Benchmark Applications used in the Evaluation

#	Applic.	No. of Functions	% Exec. Covered	% Comm. Covered
1	Barnes	38	99.19	99.45
2	Fmm	70	91.86	84.12
3	Raytrace	85	75.32	77.55
4	Clustalw	89	96.06	86.17
5	KLT	103	99.85	98.73

weights for filtering the top kernel functions in the application. Table 1 provides detailed specifications of the five applications. Column 3 provides the number of functions in each application. As it can be seen in Column 4 and 5, considering a subset of the top contributing functions covers most of the heavily executing and heavily communicating functions in these applications. The results of the absolute ranking are provided in Figure 3. The vertical lines show the range of the cost of solutions found by BF for each application. It can be seen that HS is able to find close-to-optimal solutions. The costs of solutions found by SA, TS and ES are also marked for comparison. Overall, SA outperforms all the other algorithms, as it is able to find close to optimal solutions.

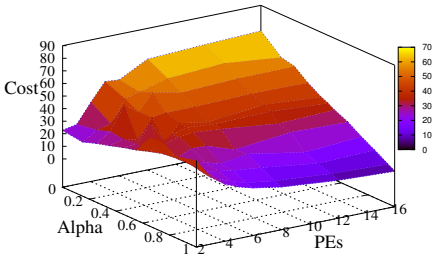

Fig. 4. Effect of variation of α and No. of PEs on cost of the partitions found by tabu search for the *Raytrace* application from SPLASH-2 benchmark suite.

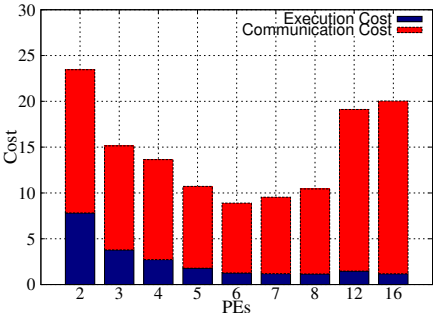

Fig. 5. Variation of execution and communication cost of Clustalw application. Minimum cost values correspond to the optimal number of PEs (at $PEs = 6$) for the given α and β.

In the above experiments, we have kept α and β fixed. Figure 4 presents the variation of the costs of partitions found by Tabu Search algorithm for *Raytrace* application from SPLASH-2 benchmark suite. It can be seen in this figure that for lower values of α (higher values of β) increasing more PEs, increases cost because of the increased communication among PEs. Higher values of α (lower values of β) implies that the communication is not expensive on the platform at hand and adding more PEs will decrease the total cost of partition. Similar

analysis can also be performed for various values of α and β and the number of PEs for other applications.

Another important result which can be obtained from this evaluation is the optimal number of PEs required for a given application for a given platform (values of α and β). Adding more PEs may reduce the execution time of an application by performing the job in parallel, however, the disadvantage is the increase in data communication among the PEs, which may kill the anticipated speedup. This analysis can especially be important for platforms where number of PEs can be reconfigured. For instance, in Molen architecture [15], the number of cores can vary from 2-5 with an upper limit imposed by the resources on FPGA. Even for non-reconfigurable platforms, where number of cores cannot be altered, the additional PEs can be switched-off to save the energy consumption.

In order to perform the analysis for optimal number of PEs, we have modified the equation to calculate the total cost of the *partition* to consider the effect of the addition of PEs as follows:

$$TC = \alpha(EC_s + EC_p/N) + \beta CC \; ; \; 0 \leq \alpha \leq 1, 0 \leq \beta \leq 1 \; and \; \alpha + \beta = 1, \quad (2)$$

where EC_s is the execution cost of the serial region of the application, EC_p is the execution cost of the parallel region of the application and N is the number of PEs. In this simple equation, we have assumed that the cost of the parallel region of the application scales with the number of the available PEs. Furthermore, we did not consider the overhead of parallelization, etc, in this analysis, as the main objective here is to show the effect of increasing the number of PEs on the execution and the communication costs of a *partition* for an application for certain values of α and β. Figure 5 provides a plot of the execution and the communication cost of the *Clustalw* application for a number of values of PEs. For this graph, we have given equal weights to α and β. It can be observed from this plot that increasing the number of PEs reduces the execution cost, but the interaction among PEs in the form of data communication increases resulting in an increase in the communication cost. The optimal number of PEs corresponds to the minimal cost values, which in this case happens when the number of PEs is equal to 6.

7 Conclusions

A key factor that substantially affects the performance of heterogeneous multicore architectures is the scheme used to partition an application. Hence, a proper mechanism to evaluate various partitioning schemes is important. In this work, we presented a methodology to evaluate application *partitions* for multicore architectures. Functions are assumed as vertices in a graph and the communication among functions is represented by the weights of the edges connecting them. We presented PET— a flexible and modular partition evaluation tool— implementing the proposed methodology.

We provided the detailed relative and absolute evaluations of a heuristic task clustering algorithm, as a case study to prove the effectiveness of our methodology. From the evaluation, it was concluded that the heuristic algorithm is able to

find close to optimal solutions in very short time compared to other algorithms selected in this comparison. In our future work, we plan to add the profiles of applications from various domains in our tool. Furthermore, benchmark suites can be added to broaden the choices for input set specifications. Utilization of Pareto based multi-objective application partitioning evaluation can also be investigated to provide the whole Pareto-front.

Acknowledgements. This research is partially supported by the Artemisia iFEST project (grant 100203), the Artemisia SMECY project (grant 100230), the FP7 Reflect project (grant 248976) and the Higher Education Commission (HEC) Pakistan.

References

1. Buchty, R., et al.: A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators. Concurrency and Computation: Practice and Experience 24(7), 663–675 (2012)
2. Martin, G.: Overview of the mpsoc design challenge. In: DAC, pp. 274–279 (2006)
3. Haritan, E., et al.: Multicore design is the challenge! what is the solution? In: DAC, pp. 128–130 (June 2008)
4. Tahaee, S.A., et al.: A polynomial algorithm for partitioning problems. ACM Trans. Embed.Comput. Syst. 9(4), 34:1–34:38 (2010)
5. López-Vallejo, M., et al.: On the hardware-software partitioning problem: System modeling and partitioning techniques. ACM Trans. Des. Autom. Electron. Syst. 8(3), 269–297 (2003), http://doi.acm.org/10.1145/785411.785412
6. Ostadzadeh, S.A., Meeuws, R., Ashraf, I., Galuzzi, C., Bertels, K.: The Q^2 Profiling Framework: Driving Application Mapping for Heterogeneous Reconfigurable Platforms. In: Choy, O.C.S., Cheung, R.C.C., Athanas, P., Sano, K. (eds.) ARC 2012. LNCS, vol. 7199, pp. 76–88. Springer, Heidelberg (2012)
7. Ostadzadeh, S.: Quantitative Application Data Flow Characterization for Heterogeneous Multicore Architectures. Ph.D. thesis, TU Delft, Netherlands (December 2012)
8. Sairaman, V.: A generalized framework for automatic code partitioning and generation in distributed systems. Graduate School Theses (June 2010)
9. Leland, R., et al.: An empirical study of static load balancing algorithms. In: Proceedings of the Scalable High-Performance Computing Conference, pp. 682–685 (May 1994)
10. The chaco user's guide, version 1.0. Tech. rep. (October 1993), http://www.cs.sandia.gov/~bahendr/chaco.html
11. Schloegel, K., et al.: A new algorithm for multi-objective graph partitioning. In: Proceedings of Europar, pp. 322–331. Springer (1999)
12. Singh, J.P., et al.: Splash: Stanford parallel applications for shared-memory. SIGARCH Comput. Archit. News 20(1), 5–44 (1992)
13. Thompson, J.D., et al.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Research (1994)
14. Lucas, B.D., et al.: An Iterative Image Registration Technique with an Application to Stereo Vision, pp. 674–679 (1981)
15. Vassiliadis, S., et al.: The MOLEN Polymorphic Processor. IEEE Transactions on Computers 53(11), 1363–1375 (2004)