Available online at www.sciencedirect.com
SCIENCE DIRECT
J. Parallel Distrib. Comput. 66 (2006) 77–98
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems☆

Yu-Kwong Kwok[a], Anthony A. Maciejewski[b], Howard Jay Siegel[b,c], Ishfaq Ahmad[d,*], Arif Ghafoor[e]

[a]*Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong, Hong Kong*
[b]*Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523-1373, USA*
[c]*Department of Computer Science, Colorado State University, Fort Collins, CO 80523-1873, USA*
[d]*Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019-0015, USA*
[e]*School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA*

Received 17 February 2005; received in revised form 2 June 2005; accepted 23 June 2005
Available online 10 November 2005

Abstract

Minimization of the execution time of an iterative application in a heterogeneous parallel computing environment requires an appropriate mapping scheme for matching and scheduling the subtasks of a given application onto the processors. Often, some of the characteristics of the application subtasks are unknown a priori or change from iteration to iteration during execution-time based on the inputs being processed. In such a scenario, it may not be feasible to use the same off-line-derived mapping for each iteration of the application. One possibility is to employ a *semi-static* methodology that starts with an initial mapping but dynamically performs remapping between application iterations by observing the effects of the changing characteristics of the application's input data, called *dynamic parameters*, on the application's execution time. A contribution in this paper is to implement and evaluate a semi-static methodology involving the on-line use of off-line-derived mappings. The off-line phase is based on a genetic algorithm (GA) to generate high-quality mappings for a range of values for the dynamic parameters. A dynamic parameter space partitioning and sampling scheme is proposed that partitions the parameter space into a number of *hyper-rectangles*, within which the "best" mapping for each hyper-rectangle is stored in a mapping table. During the on-line phase, the actual dynamic parameters are observed and the off-line-derived mapping table is referenced to choose the most suitable mapping. Experimental results indicate that the semi-static approach outperforms a dynamic on-line approach and performs reasonably close to an infeasible on-line GA approach. Furthermore, the semi-static approach considerably outperforms the method of using the same mapping for all iterations.

Keywords: Heterogeneous computing; Parallel processing; Mapping; Scheduling; Iterative task graphs; Genetic algorithms; Automatic target recognition

1. Introduction

1.1. On-line use of off-line-derived mappings

Heterogeneous computing (HC) encompasses a great variety of situations (e.g., see [19,24,28,35,46,48,53,54]). This paper focuses on a particular application domain in which (1) an iterative application is to be mapped onto an associated specific type of dedicated heterogeneous parallel hardware platform and (2) the execution of each iteration can be represented by a directed acyclic graph (DAG) of

☆ This research was jointly supported by the Hong Kong Research Grants Council (under Contract No. HKU 7124/99E), by the DARPA/ITO Quorum Program (under NPS Subcontract Nos. N62271-98-M-0217 and N62271-98-M-0448, and GSA Contract No. GS09K99BH0250), by the DARPA/ITO Quorum Program through the Office of Naval Research (under Grant No. N00014-00-1-0599), and by the Colorado State University Abell Endowment.
* Corresponding author. Fax: +1 817 272 3784.
E-mail addresses: ykwok@eee.hku.hk (Yu-Kwong Kwok), aam@colostate.edu (A.A. Maciejewski), hj@colostate.edu (H.J. Siegel), iahmad@cse.uta.edu (I. Ahmad), ghafoor@ecn.purdue.edu (A. Ghafoor).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.06.015

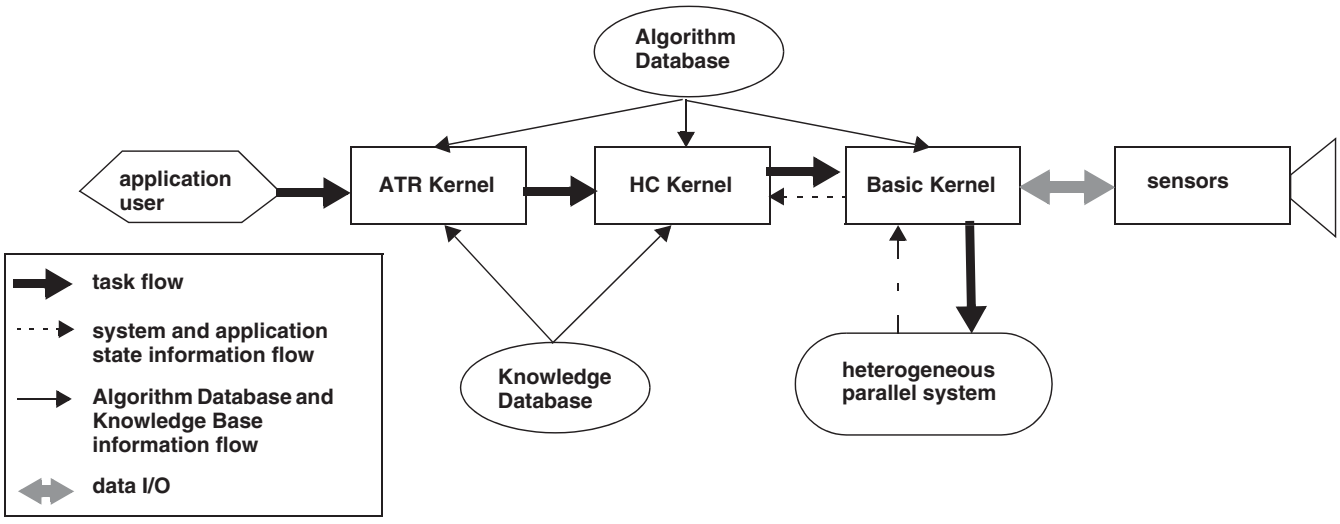


Fig. 1. Conceptual design of an operating system for ATR applications.

subtasks. To minimize the execution time of such an iterative application in a heterogeneous parallel computing environment, an appropriate mapping scheme is needed for matching and scheduling the subtasks onto the processors [1,2,5–8,29,39,34,55]. However, when some of the characteristics of the application subtasks are unknown a priori and will change from iteration to iteration during execution-time based on the inputs being processed, it may not be feasible or desirable to use the same off-line-derived mapping for each iteration of the application.

An example of such a problem domain are iterative automatic target recognition (ATR) tasks, where a sequence of images are received from a group of sensors and various image-processing operations are required to generate a real-time scene description. Most notable ATR-based applications are homeland security imaging [50] and medical monitoring systems [32]. In ATR, the characteristics of a subtask's input data, such as the amount of clutter and the number of objects to be identified, varies from image to image, and thus, may lead to large variations in the subtask's processing requirements from one iteration to the next.

In such situations, a semi-static methodology [9,10] can be employed, which starts with an initial mapping but dynamically decides whether to remap the application with a mapping previously determined off-line (i.e., statically). This can be done by observing, from one iteration to another, the effects of the changing characteristics of the application's input data, called dynamic parameters, on the application's execution time. In other words, the operating system will be able to make a heuristically determined decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system will be able to select a pre-computed and stored mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking). This remapping process will, in general, require a certain system reconfiguration time for relocating the data and program modules. The semi-static method differs considerably from other real-time HC mapping techniques in that it involves the on-line real-time use of off-line precomputed mappings. This is significant because it is possible for off-line heuristics to have much longer execution times to search for a good solution (mapping) than what is practical for an on-line heuristic. Thus, with the semi-static method, the mapping quality of a time-consuming off-line heuristic can be approached at real-time speeds. The focus of this paper is the evaluation of a method for performing semi-static mappings.

A previously proposed conceptual design of a high-level operating system for ATR applications, which includes the capability for the on-line use of off-line computed mappings, is depicted in Fig. 1 [9,10]. This conceptual design has its roots in the high-level model presented in [14] for automatic dynamic processor allocation in a partitionable parallel machine with homogeneous processors (called PASM [47,49]).

The ATR Kernel in Fig. 1 makes decisions on how a given ATR application task should be accomplished, including determining the partial ordering of subtasks and which algorithms should be used to accomplish each subtask. The HC Kernel uses a semi-static method to decide how the partially ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. A subtask may have a data-parallel implementation, and, thus, may be assigned to a set of processors. Also, the HC Kernel interacts with the Basic Kernel (the low-level operating system) to execute the application and monitors its execution so it can decide at the end of each iteration through the application if the subtasks should be remapped onto the hardware platform. Thus, the ATR Kernel deals with application issues, while the HC Kernel deals with implementation issues. Information from the Algorithm Database and

the Knowledge Base is used to support the ATR and HC Kernels.

The design of the ATR Kernel and HC Kernel was the focus of Budenske et al. [10]. In addition, two methods were suggested for determining the representative choices for the dynamic parameter values to use for each static mapping. The first was to have the application developer specify what these representative values should be. The second was to have the application developer specify a range for each dynamic parameter, and then the representative values are uniformly distributed over that range. Neither of these methods was ever evaluated in any way, as this was not the focus of Budenske et al. [10]. In particular, the performance of off-line mappings based on points selected in either of these ways was never determined. In contrast, our current work focuses just on the semi-static mapping approach. The current research does not depend on an application developer to guess at what might be appropriate representative values; nor does it take the simplistic approach of selecting representative values uniformly across the range. Instead, we derive a procedure for statistically sampling each region of the multidimensional dynamic parameter space to find a mapping that is near optimal for the range of parameters within that region. We evaluate the current approach in a variety of ways, including comparing it to the simplistic uniform distribution method of Budenske et al. [10].

A more detailed discussion of the ATR problem domain is presented in Section 1.2. Examples of other applications whose characteristics are similar to those of the iterative ATR applications and platforms include sensor-based robotics, intelligent vehicle highway systems, air traffic control, nuclear facility maintenance, weather prediction, intruder detection, and manufacturing inspection. The performance of the semi-static method in comparison to other methods will depend upon the exact application and exact platform under consideration.

The application to be mapped is iterative in structure and each iteration is modeled by a DAG in which the nodes represent subtasks and the edges represent the communications among subtasks. The model used for an application task is described in Section 3. The attributes associated with the DAG, such as the computation time of a subtask and the communication time between subtasks, are modeled by equations that are functions of the dynamic parameters. Examples of dynamic parameters include the contrast level of an image, the number of objects in a scene, and the average size of an object in a scene. Thus, as the dynamic parameters change from one iteration (one image) to the next iteration, the mapping currently in use may not be suitable and a remapping of the subtasks onto the processors may need to be performed. However, performing a remapping requires a certain system reconfiguration time. Given the current mapping, a new mapping, and the system estimated reconfiguration time, the HC Kernel has to decide whether a remapping is to be done. This framework can be applied to any task graph structure represented as a DAG.

1.2. An example application domain: ATR

Simply stated, an ATR system takes a set of images iteratively from a group of sensors and produces some type of description of the scene [56]. The most notable real-life examples of an ATR-based system are homeland security imaging [50] and medical monitoring systems [32]. A simplified example of an ATR task for tracking forwardlooking ladar infra-red (FLIR) images is shown in Fig. 2 [59]. The various types of image processing elements required in an ATR system can be broadly classified into three groups: low-level processing (numeric computation), intermediate-level processing (quasi-symbolic computation, e.g., where numeric and symbolic types of operations are used to describe surfaces and shapes of objects in the scene), and high-level processing (symbolic computation, e.g., for producing the scene description) [3,4,13,15,20,25,27]. Heterogeneous parallel architectures are appropriate computing platforms for efficiently handling computational tasks with such diverse requirements.

A key technical issue that must be addressed to exploit the inherent potential of heterogeneous parallel computing systems to efficiently execute ATR applications is the development of a high-level operating system that can fully utilize the architectural flexibility of such a system. Such a high-level operating system must be able to assign each ATR application subtask to the processors where it is best suited for execution. Often, subtasks can execute concurrently, sharing resources. Because the execution time of application subtasks in an ATR system is highly input-data dependent (e.g., number of currently located objects), this matching and scheduling of application subtasks to processors must be performed dynamically at execution-time for best performance.

The semi-static concept was proposed in [10] for a class of ATR applications, where each application can be modeled as an iterative execution of a set of partially ordered subtasks. The ATR applications in this class are production jobs that are executed repeatedly and, hence, it is worthwhile to invest off-line time to determine an effective mapping of such an application onto the hardware platform used to execute it. The automatic target acquisition (ATA) system described in [16] and the tracking system described in [17] are examples of such iterative ATR applications.

1.3. Contributions of this research

The objective of this paper is to implement and evaluate a semi-static methodology, called the on-line use of off-line-derived mappings (denoted as On–Off in subsequent sections), whose conceptual structure was proposed in [10]. In particular, the goal of the present study is two-fold: (1) to design novel and practical off-line mapping generation methods; and (2) to evaluate the ideas underlying the design and use of the On–Off method for a specific class of

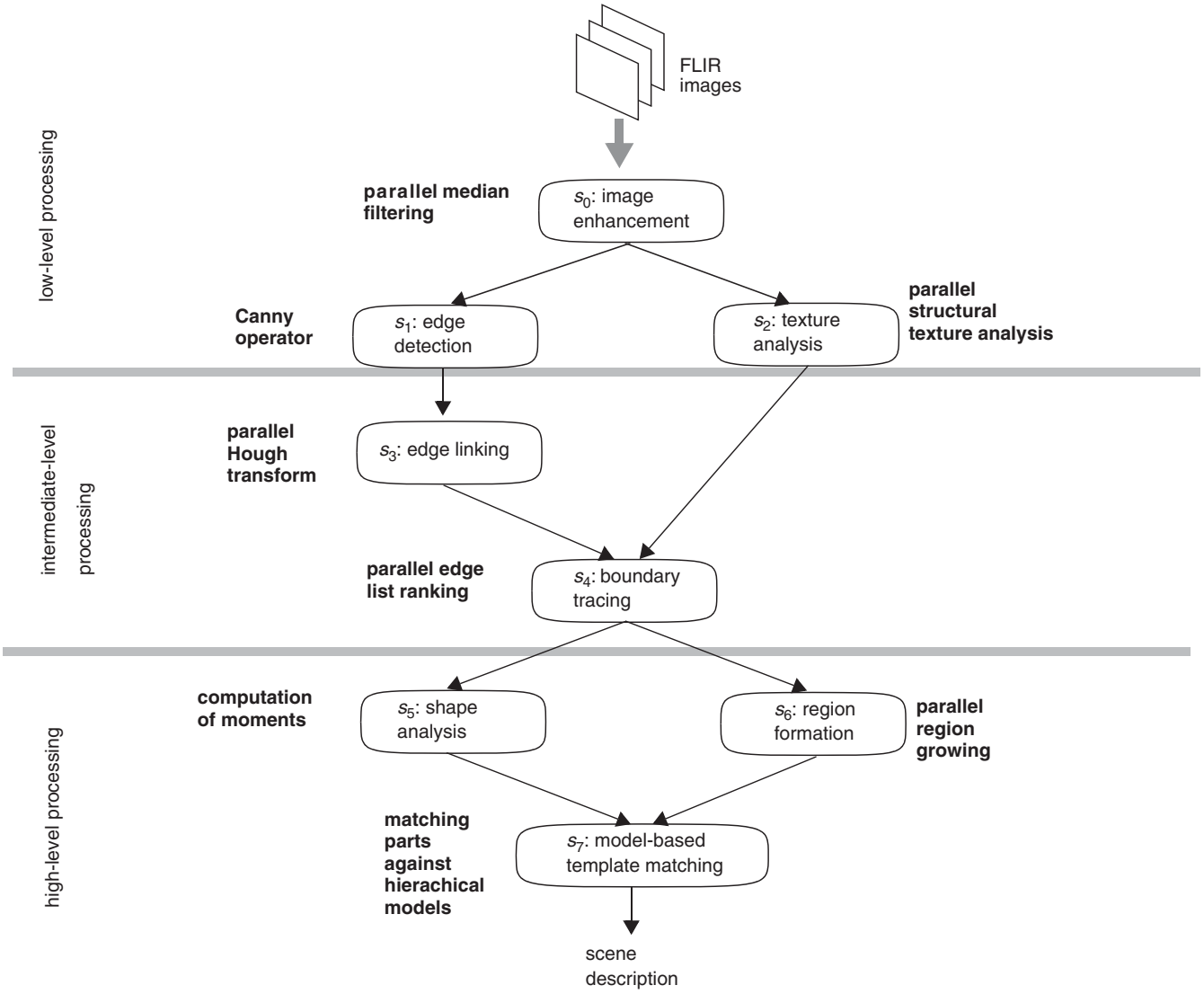


Fig. 2. A directed acyclic graph modeling an ATR application.

computational structures and hardware platforms. This is done by simulating the methodology and studying its behavior for various situations.

The implementation of the On–Off methodology entails addressing the fundamental research issue of how to select representative mappings off-line for on-line use. To solve this problem, a novel dynamic parameter space partitioning and sampling scheme is proposed in Section 4. During the off-line phase, a genetic algorithm (GA) is used to generate high-quality mappings for a range of values for the dynamic parameters. Specifically, the dynamic parameter space is partitioned into a number of hyper-rectangles, within which the "best" mapping for each hyper-rectangle is stored in a mapping table. During the on-line phase, the actual dynamic parameters are observed and the off-line-derived mapping table is referenced to choose the most suitable mapping. Experimental results, presented in Section 5, indicate that this semi-static approach is effective in that it consistently gave performance that was comparable to that of using the same GA on-line with the exact dynamic parameter values for the *next* iteration (which is physically impossible). Also, the semi-static approach considerably outperforms using the same mapping for all iterations and outperforms a representative dynamic on-line mapping heuristic.

2. Related work

In [38], a greedy policy is suggested for the dynamic remapping of iterative data parallel applications, such as fluid dynamics problems, on a homogeneous message-passing architecture. In these types of applications, multiple processors work independently on different regions of the data domain during one iteration. As the focus of computation may shift from one region to another between iterations, some processors will be idle while some other processors

will be overloaded if the initial mapping of workload is not changed. Thus, remapping is useful for balancing the workload across the processors and thus, reducing the execution time of an iteration. The technique reported in [38] works by monitoring from iteration to iteration a function that indicates the average processor idle time since the last remapping. Once it is detected that a local minimum point of this function is reached at a certain iteration, a remapping is done. This approach is not applicable to the scenario considered in this work for two reasons. First, it assumes a homogeneous processing environment rather than a heterogeneous one. Second, it is designed for independent data parallel computations such that a task does not have the subtask precedence constraints that are modeled as DAGs here.

A framework has been recently suggested for determining an off-line mapping of image processing applications, modeled as a linear task chain, to homogeneous distributed-memory machines [31]. The mapping technique, which is based on the shortest path algorithm, can only be applied to a linear task chain. The goal is to produce a static mapping that has good average performance without regard to the actual variations of task parameters during execution time. This is in contrast to the semi-static technique described here that adjusts the mapping during execution time according to the changes in dynamic parameters. The algorithm used in [31] is not applicable to the scenario considered here because tasks are represented as DAGs, rather than chains, and the hardware target is assumed to be heterogeneous rather than homogeneous.

A scheduling and allocation scheme reported in [41] for regular scientific applications uses a model that is similar to the one used here in that a subtask in an application is a data parallel program. According to this model, a regularly structured high-performance fortran (HPF) application is decomposed into a DAG in which the functional parallelism is captured by the DAG structure but the data parallelism is encapsulated within each subtask. An approximate algorithm is used for mapping the application to a distributed-memory machine. However, the approach is static in that the mapping algorithm is applied based on cost estimates at compile-time and no execution-time scheme is used for tuning the scheduling and allocation. Furthermore, a homogeneous target platform is assumed.

In [60], a run-time support technique is proposed for minimizing the execution time of some iterative computations on message-passing architectures by using the loop unfolding method. The technique used is essentially based on exploiting the inter-iteration parallelism to further minimize the execution time. Similar to the above approaches, the mapping is only done statically based on cost estimates and no execution-time adjustment of mapping is done, and a homogeneous target platform is assumed.

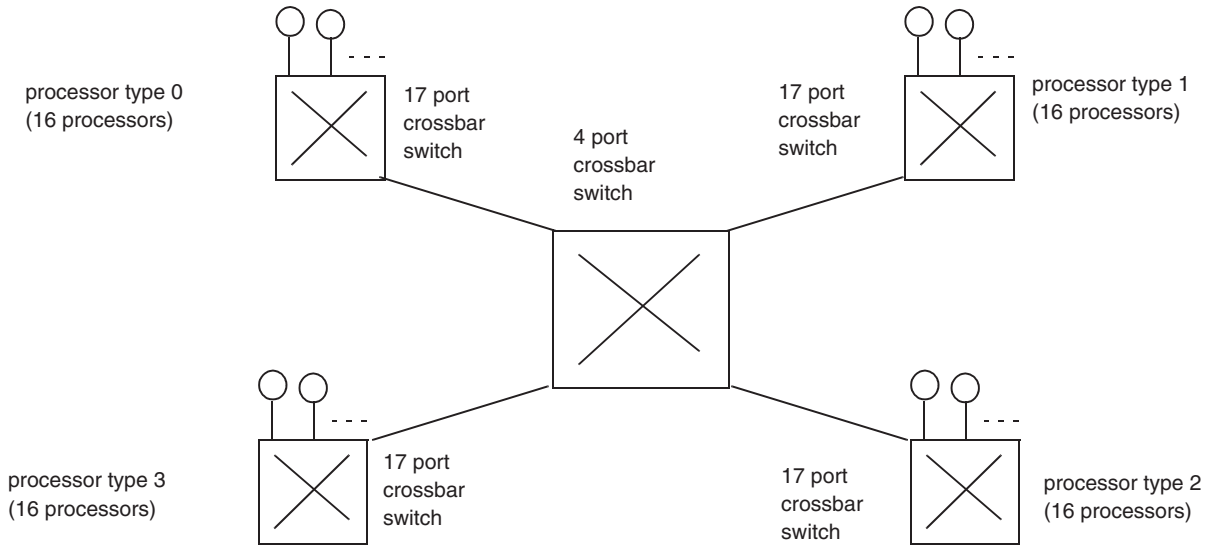
In [52], an automatic machine selection scheme is proposed for allocating application tasks to a network of autonomous computers. The major focus is to minimize network congestion by taking into account the application's communication topology. Unlike the task-mapping algorithms considered in this paper, execution times of the applications are not explicitly optimized. Several scientific applications (e.g., 2D fast-Fourier transform) are tested using an experimental system implemented using the Remos API [42]. In [45], the dynamic data redistribution costs of the 2D Jacobi application are analytically modeled by exploiting the application's structure under the AppLeS (Application Level Scheduling) framework. The model is rather specific and can accurately predict the redistribution costs of the 2D Jacobi application under different simulated scenarios. This is very different from the generalized task execution model used in the study presented here.

In [43], an integrated compile-time and run-time scheme for predicting the data redistribution cost for multi-phase parallel applications on a software distributed shared memory system is suggested. The crux of the scheme is a clever exploitation of the knowledge about the data access patterns, page locations, and the distributed shared memory protocol. Again, the focus is mainly on execution costs related to data placements but not the overall execution times. In [37], an integrated load balancing and scheduling scheme is proposed for efficient execution of parallel applications on a network of autonomous machines. The proposed scheme exploits the synergy among the compile-time analysis mechanism, run-time system management, and OS level load balancing facilities. Unlike the general HC platform considered here, the target platform is a loosely coupled homogeneous machines. On a network of four machines, each of which is a four-CPU SMP, the integrated scheme can efficiently execute several scientific applications (e.g., Jacobi, matrix multiplication, etc.).

3. System model

To evaluate the On–Off semi-static mapping methodology, a particular sample architecture is chosen; however, the On–Off method can be adopted for other target architectures. The sample target HC platform considered here is based on the expected needs of ATR applications that are of interest to the US Army Research Laboratory (e.g., [16,17]). Specifically, it contains four different types of processors (e.g., [18,30]), with 16 processors of each type (see Fig. 3(a)).

The processors are connected via crossbar switches in such a way that each processor has exactly one input port and one output port. Communications among processors of the same type are assumed to be symmetric in the sense that the conflict-free time for any pair of processors (of the same type) to communicate is the same (see Fig. 3(b)). For simplicity, it is assumed that if a data-parallel implementation of a given subtask uses a virtual machine of processors, all processors will be of the same type. Given this and the symmetry property of the inter-processor communications among processors of the same type, the expected execution



(a)

	type 0		type 1		type 2		type 3	
	S	1/*R*	*S*	1/*R*	*S*	1/*R*	*S*	1/*R*
type 0	0.92	0.41	9.04	3.72	10.47	5.58	22.47	11.16
type 1	9.04	3.72	0.83	0.37	4.62	4.40	7.00	7.44
type 2	10.47	5.58	4.62	4.40	0.86	0.45	11.37	6.38
type 3	22.47	11.16	7.00	7.44	11.37	6.38	0.83	0.53

(b)

Fig. 3. (a) The target heterogeneous computing platform consisting of four types of processors with 16 in each type; (b) message start-up times, S (in ms), and transmission times per unit data, $1/R$, of the inter-processor communication channels.

time of a particular multiprocessor implementation of a subtask is independent of which fixed-size subset of the processors of a given type are assigned to execute the subtask. It is assumed that one processor in each virtual machine is responsible for data communication to other virtual machines. This assumption simplifies the simulation study, but is not required for the On–Off methodology. In addition, the hardware platform includes (1) a workstation, for off-line operations to develop an application implementation and for use as the application user interface, and (2) a host processor, which monitors the application implementation during its execution and implements the on-line HC Kernel. It is assumed that when a given ATR application is executing on a platform, that platform is dedicated to that application. It should be noted that the On–Off strategy is applicable to a wide variety of computing platforms that vary from the test platform described.

An application task is modeled as a DAG, with n nodes representing subtasks s_i $(0 \leqslant i \leqslant n-1)$ and e edges representing inter-subtask communications. To illustrate the On–Off semi-static mapping approach, a simplified model is used for subtask execution time and inter-subtask communication time. However, the On–Off framework does not depend on how the subtask execution time and inter-subtask communication time are modeled. The way in which the four dynamic parameters α, β, γ, and μ are used in each subtask execution time (and inter-subtask communication time) equation given below is a simple approach for the purpose of illustration in the simulation studies to be described. In reality, a single-dynamic parameter can impact any subset of the components of a given subtask's execution time equation. Furthermore, the way in which a particular dynamic parameter impacts a given component of a subtask's execution time may not be linear (as assumed here), and may differ for different subtasks. Clearly, the structure and details of the execution time complexity equations for subtasks, as a function of dynamic parameters, is task dependent and can vary greatly. The determination of these equations is considered the responsibility of the application developer [9,10] and outside the scope of this paper. These equations are just an input to the

Table 1
Definition of notation

Notation	Definition
s_i	Subtask i
n	Number of subtasks in the application task
Subtask execution time equation variables	
$t_u(s_i)$	Execution time of subtask s_i on virtual machine u
h_{iu}	Heterogeneity factor indicating the relative speed of the ith subtask on machine u
a_i	Coefficient for the parallel portion of subtask i's execution time
b_i	Coefficient for the parallelization overhead portion of subtask i's execution time
c_i	Coefficient for the serial portion of subtask i's execution time
p	Number of processors used in a virtual machine
Inter-subtask communication time equation variables	
d_{ij}	Size of the fixed portion of the data to be transferred between subtasks s_i and s_j
e_{ij}	Coefficient for the size of the variable portion of data to be transferred between s_i and s_j
C_{uv}	Communication time between virtual machines u and v
S_{uv}	Message start-up time between virtual machines u and v
R_{uv}	Data-transfer rate between virtual machines u and v
Dynamic parameters	
α	Dynamic parameter impacting the parallel workload
β	Dynamic parameter impacting the parallelization overhead
γ	Dynamic parameter impacting the serial workload
μ	Dynamic parameter impacting the size of the variable portion of data transfer

On–Off methodology presented here, and when considering the results of the paper the reader should note that the execution time and communication time equations are parts of a simplified model. Table 1 summarizes the notation to be used throughout the paper.

The simple execution time expression used in this model is a version of Amdahl's law extended by a term representing the parallelization overhead (e.g., synchronization and communication). The serial and parallel fractions of a subtask are frequently represented using similar models (e.g., [11,21,36,44]). The execution time expression for subtask s_i includes: (a) three dynamic parameters, α, β, and γ, (b) the number of processors used, p, and (c) three coefficients a_i, b_i, and c_i. The parallel fraction and serial fraction of subtask s_i are represented by $a_i\alpha/p$ and $c_i\gamma$, respectively. The parallelization overhead is represented by $b_i\beta\log p$ and h_{iu} is the heterogeneity factor, indicating the relative speed of the subtask s_i on the type of processor used in virtual machine u. The heterogeneity factor reflects the real-world differences among machines that could impact the execution time of a given subtask on a given machine. These factors include, but are not limited to, CPU clock rate, the number of levels of cache and sizes of cache at each level, the exact instruction set and execution time for each instruction, and the pipeline structure. The impact of each of such factors will depend on the precise code for the given subtask. Thus, in general, different subtasks will have affinities for different machines. The execution time $t_u(s_i)$ of subtask s_i on virtual machine u is modeled by the expression

$$t_u(s_i) = h_{iu} \cdot (a_i\alpha/p + b_i\beta\log p + c_i\gamma). \quad (1)$$

By differentiating this equation and equating the derivative to zero, the optimal value of p that leads to the minimum execution time for a given subtask is $p_{\text{opt}} = (a_i\alpha)/(b_i\beta)$. The mapping heuristic will not assign more processors than a subtask's p_{opt}.

It is assumed that the size of the data set to be transferred between two subtasks s_i and s_j consists of a fixed portion and a variable portion. The size of the fixed portion is modeled by a constant d_{ij} (independent of the input data of the application). The size of the variable portion is modeled by the product of a coefficient e_{ij} and a dynamic parameter μ. For communication between virtual machines u and v, S_{uv} and R_{uv} are the message start-up time and the data transmission rate, respectively [26] (see Fig. 3(b) for values of S_{uv} and R_{uv} based on [17,18,26]). Thus, the inter-subtask communication time between subtask s_i on virtual machine u and subtask s_j on virtual machine v is C_{uv}, which is given by

$$C_{uv}(s_i, s_j) = S_{uv} + (d_{ij} + e_{ij}\mu)/R_{uv}. \quad (2)$$

4. The semi-static mapping approach

4.1. Remapping approaches

Consider two approaches for remapping application tasks to processors during execution time (between iterations through the DAG):

- *Dynamic mapping*: Based on the current values of dynamic parameters, compute a new mapping in real time using a low complexity algorithm.

- *On-line use of off-line-derived mappings*: For each dynamic parameter, some representative values are chosen so that a number of possible scenarios are generated. Using an off-line (i.e., static) heuristic, high-quality mappings for the scenarios are precomputed and stored in a table. During execution of the application, the mapping corresponding to the scenario with values of dynamic parameters *closest to* the actual values is selected from the table to be a possible new mapping [5].

Because a static mapping heuristic (e.g., the GA used in this study) can potentially generate solutions of much higher quality than a dynamic mapping algorithm, it is interesting to investigate how well the approach of on-line use of off-line-derived mappings (using the GA) performs. Notice that even off-line generation of optimal mappings is infeasible because the heterogeneous mapping problem is NP-complete [22,23], and, thus, exponential time is needed for finding optimal solutions.

4.2. Generation of off-line-derived mappings

In the On–Off semi-static mapping approach, it is assumed that the ranges of the dynamic parameters are known. This assumption is justified because, for example, for a particular size of image, the maximum possible number of objects of a given type at a given distance is bounded and can be estimated. Once specified by the application developer, the space of dynamic parameters is partitioned into a number of disjoint regions. Formally, suppose the minima and total range sizes of the dynamic parameters are given by $\alpha_{\min}, \beta_{\min}, \gamma_{\min}, \mu_{\min}, \alpha_{\text{range}}, \beta_{\text{range}}, \gamma_{\text{range}}$, and μ_{range}, respectively. The parameter space $\Re$ can be partitioned into K^4 uniform sized disjoint regions as follows:

$$\Re(i, j, k, l) = \{(\alpha, \beta, \gamma, \mu)\}, \quad 0 \leqslant i, j, k, l \leqslant K-1, \quad (3)$$

where $\alpha_{\min} + i\alpha_{\text{range}}/K \leqslant \alpha < \alpha_{\min} + (i+1)\alpha_{\text{range}}/K$ and the ranges for β, γ, and μ are defined analogously.

Within each region (defined by specifying values for indices i, j, k, l), N dynamic parameter vectors, each composed of $(\alpha, \beta, \gamma, \mu)$, are randomly chosen. An off-line (static) heuristic is then applied to determine the mappings for these sample scenarios represented by different dynamic parameter vectors. For a random sample vector v_x $(0 \leqslant x \leqslant N-1)$, denote the corresponding mapping by M_x. The mapping M_x is then evaluated for every one of the $N-1$ other sample scenarios in the region by applying the mapping M_x to the DAG and computing the total completion time for one iteration of the application given the dynamic parameter values that define that scenario. This is repeated for each M_x. That is, the task completion times $t(M_x(v_y))$ for all x and y $(0 \leqslant x, y \leqslant N-1)$ are computed. Then, the average completion time for each mapping M_x is computed as $[\sum_{y=0}^{N-1} t(M_x(v_y))]/N$. The mapping M_x that gives the minimum of these average completion times is chosen as the representative mapping for the corresponding region in the dynamic parameter space. This representative mapping and the corresponding average completion time are stored in the off-line mapping table, which is a multi-dimensional array indexed by i, j, k, l.

This new approach for determining representative mappings differs considerably from that used in earlier work, where it was assumed that the application developer will provide the specific set of dynamic parameter values to be used to derive each representative mapping [10]. Fig. 4 depicts the block diagram of the envisioned system for determining the off-line mapping for a given region of the dynamic parameter space.

Here, the parameter space is uniformly partitioned with each parameter equally subdivided. However, different values of K can be used for each individual dynamic parameter, depending upon the specifications given by the application developer. In addition, one could use non-uniform-sized regions for the parameter space if good performance is not being achieved. In particular, if the average completion time for the representative mapping selected for a region is significantly greater than $t(M_x(v_x))$ for most values of x, then that region can be subdivided and a representative mapping determined for each sub-region. This process can be repeated recursively.

4.3. On-line retrieval of off-line-derived mappings

The input to the simulated on-line module consists of an execution profile that comprises a certain number of iterations of executing the task graph. Examples of execution profiles containing 20 iterations are shown in Table 2 in Section 5. In each profile, the dynamic parameter values change from one iteration to another. Specifically, row i represents the values of the dynamic parameters *observed* after execution of the graph for iteration i is finished. Thus, when execution of the task for iteration i begins, the on-line module does not know the (simulated) *actual* values of the dynamic parameters for that iteration. The on-line module has to determine a mapping for iteration i based on the dynamic parameter values of iteration $i-1$. In the On–Off semi-static mapping approach, the representative mapping is retrieved for the region that includes the dynamic parameter values of iteration $i-1$. If the stored pre-determined execution time of the selected mapping, plus the estimated reconfiguration time, is smaller than the execution time that occurred for iteration $i-1$, a remapping is performed; otherwise, the mapping used at iteration $i-1$ will continue to be used for iteration i. The off-line mapping technique used in this study is discussed in Section 4.5.

4.4. A dynamic approach

In this paper, several mapping approaches are examined and evaluated. The first approach is a dynamic approach that uses a fast heuristic that takes a small amount of time but generates a reasonably good solution. The heuristic used is a

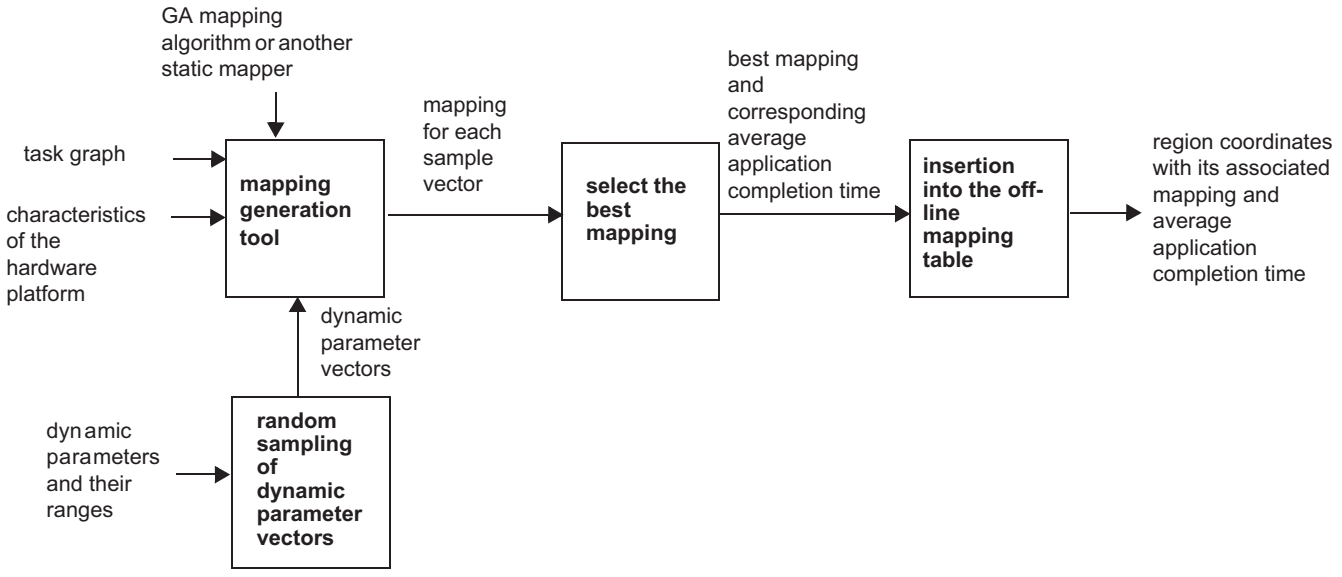


Fig. 4. Generation of the off-line mapping for a given region of the dynamic parameter space.

fast static mapping algorithm, called the earliest completion time (ECT) algorithm, that is based on the technique presented in [57] and is outlined below. It should be noted that other high-performance scheduling heuristics, such as DCP [29] and CPR [40], are unsuitable to be used for dynamic scheduling due to their high time complexity.

The ECT algorithm is used for performing dynamic mapping in a manner similar to that described in Section 4.3. The ECT algorithm is applied (in real time) to the task graph with the values for the dynamic parameters at iteration $i-1$. The resulting mapping is (with its associated estimated task execution time using the iteration $i-1$ parameters) then considered to be a potential new mapping for iteration i. Again, if the estimated gain in adopting the new mapping is greater than the reconfiguration time, the new mapping will be used in iteration i.

4.5. Genetic algorithm

GAs are a promising heuristic approach to tackling optimization problems that are intractable. There are a great variety of approaches to GAs (see [51] for a survey). The first step of designing a GA is to encode all possible solutions as chromosomes, a set of which is referred to as a population. In this study, the GA presented in [58] is extended to allow multiple processors to be assigned to a subtask. Each chromosome consists of three parts: the matching string, the processor allocation string, and the scheduling string. Each string is a vector of length n, the number of subtasks in the application task.

Let *mat* be the matching string where $mat(i) = j$ means that subtask s_i is mapped to processor type j. Let the allocation string, alloc(i), be the number of processors of type

ECT ALGORITHM:

(1) Construct a level-based list L of subtasks as follows:
  (a) Label each entry subtask (subtask that does not have predecessor) as a level 1 subtask.
  (b) For each of the remaining subtasks, label it as level i if its highest level parent is at level $i-1$.
  (c) Make L a list of subtasks sorted in ascending order of levels. For subtasks on the same level, they are sorted in descending order of the number of children. Ties are broken arbitrarily.
(2) for each subtask s_i on L do:
(3) for each processor group do:
(4) for $p = 1$ to p_{opt} do:
(5) Find the earliest start time t_{start} such that there are p available processors.
(6) Note s_i's completion time using p processors.
(7) endfor
(8) endfor
(9) Schedule s_i using p processors of the processor group (starting at the corresponding t_{start}) that minimizes its completion time.
(10) Update the availability times for the processors in this group.
(11) endfor

Table 2
Execution profiles of dynamic parameters: (a) Profile A (average percentage change in dynamic parameter values $\Delta = 5\%$) and (b) Profile B ($\Delta = 40\%$)

Iteration	(a) Profile A			
	α	β	γ	μ
0	3000	15	300	60
1	2821	15	287	63
2	2949	12	302	65
3	3073	12	286	68
4	3228	11	273	71
5	3090	13	258	67
6	3256	11	272	70
7	3424	16	259	73
8	3621	16	271	75
9	3811	13	260	78
10	4014	17	245	81
11	4229	13	257	77
12	3994	19	242	80
13	4179	15	253	83
14	4386	15	264	78
15	4208	13	249	82
16	4016	14	236	77
17	3835	16	226	81
18	4026	19	238	84
19	4258	16	251	88
20	4479	15	265	92
Iteration	(b) Profile B			
0	3000	15	300	60
1	4309	15	409	82
2	2635	7	268	43
3	3894	6	361	27
4	2241	8	197	39
5	1265	12	287	52
6	1699	16	420	75
7	1138	11	282	50
8	1543	12	153	67
9	2205	17	225	97
10	3198	10	332	51
11	4678	18	477	73
12	2588	8	315	48
13	1358	16	211	67
14	1794	17	307	98
15	2605	11	163	61
16	3719	17	240	87
17	2478	9	332	53
18	1507	16	466	76
19	2081	8	243	50
20	3053	17	149	70

j assigned to subtask s_i, where $\text{alloc}(i) \leqslant p_{\text{opt}}$ for s_i. Typically, multiple subtasks will be assigned to some of the same processors in a processor group, and then executed in a non-preemptive manner based on an ordering that obeys the precedence constraints (data dependencies) specified in the application task DAG. The scheduling string is a topological sort of the DAG (i.e., a valid total ordering of the partially ordered DAG).

In the initial population generation step, a predefined number of chromosomes are randomly created as follows. A new matching string is obtained by randomly assigning each subtask to a processor group. Scheduling strings are generated by performing a topological sort on the DAG and then applying a mutation operator to create random valid scheduling strings (i.e., schedules that are also valid topological sorts). The initial allocation string for each chromosome is generated by randomly selecting a value from 1 to p_{opt} as the number of processors allocated for each subtask (for the processor group type specified in the matching string). The solution from the ECT algorithm is also encoded as a chromosome and may be included in the initial population as a "seed." Indeed, it is common in GA applications to incorporate solutions from some non-evolutionary heuristic into the initial population, which may reduce the time needed for finding a satisfactory solution. In the GA used, it is guaranteed that the chromosomes in the initial population are distinct from each other. After the initial population is generated, the genetic operators crossover, mutation, and selection are applied one after the other for a number of generations (1000 in this study) or until the solution quality does not improve for 150 generations. Elitism is used to ensure that the best solution for each generation is explicitly protected from being modified by the mutation and crossover operations. The GA is executed ten times for a given dynamic parameter vector. To enhance diversity, only five of the ten runs include a chromosome generated by the ECT algorithm in the initial population. For more details of the GA, the reader is referred to [58].

5. Performance results

5.1. Introduction

Four approaches were compared in the experiments: (i) the On–Off approach; (ii) the ECT algorithm as a dynamic scheduling algorithm; (iii) the infeasible approach of using the GA as a real-time dynamic scheduling algorithm (referred to as GA on-line); and (iv) an ideal but impossible approach which uses the GA on-line with the exact (as yet unknown) dynamic parameters for the iteration to be executed (referred to as Ideal). The latter two schemes are included only to provide points of references for comparing the solution quality of the first two approaches.

5.2. Workload

To investigate the performance of the On–Off approach with the proposed dynamic parameter space partitioning and sampling methods, task graphs with four different structures were used. These graphs included in-tree graphs, out-tree graphs, fork-join graphs, and randomly structured graphs (see Fig. 5 for examples). Graphs with sizes 10, 50, 100, and 200 nodes were considered. For each graph structure and size, ten graphs were used in the simulation studies. Thus, a total of $4 \times 4 \times 10 = 160$ different graphs were

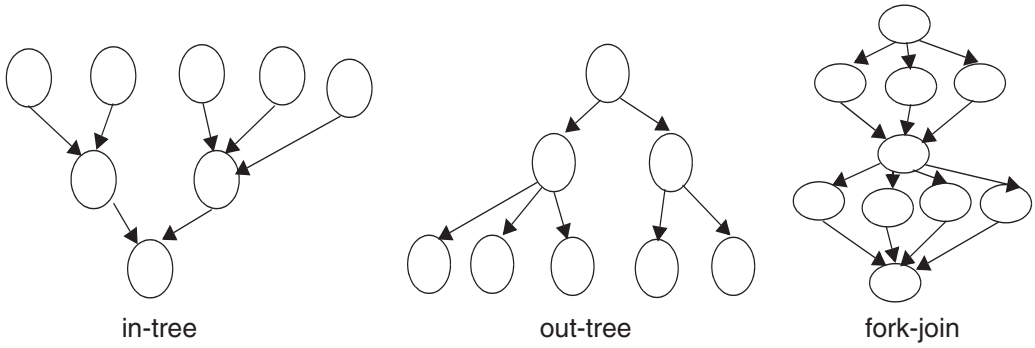


Fig. 5. Different common regular graph structures.

generated. According to [33], parallel algorithms can be classified based on the structure of their task graphs, and experience shows that most parallel algorithms belong to one of only a small number of classes. Examples of classes of task graphs are those representing asynchronous divide-and-conquer, multilevel or multiphase series-parallel, and pipelined parallel algorithms. Thus, in this study, tree structured task graphs are used to model many divide-and-conquer type of algorithms and fork-join task graphs model many multilevel or multiphase series-parallel algorithms. Randomly structured graphs were also included to represent parallel algorithms that do not fall into the other categories.

The randomly structured task graphs were generated as follows. Given the number of nodes n, the height (the number of levels) of the graph was randomly generated from a uniform distribution with range $[1 - 2\sqrt{n}]$. At each level, the number of nodes was randomly generated also using a uniform distribution but with the maximum value modified to guarantee that the final total number of nodes is n. The nodes at each level were then randomly connected to nodes at a higher level. The number of children for each node was chosen from a uniform distribution with range [0–7], but not exceeding the number of nodes specified for the next level. The regular graphs were generated according to their predefined structures.

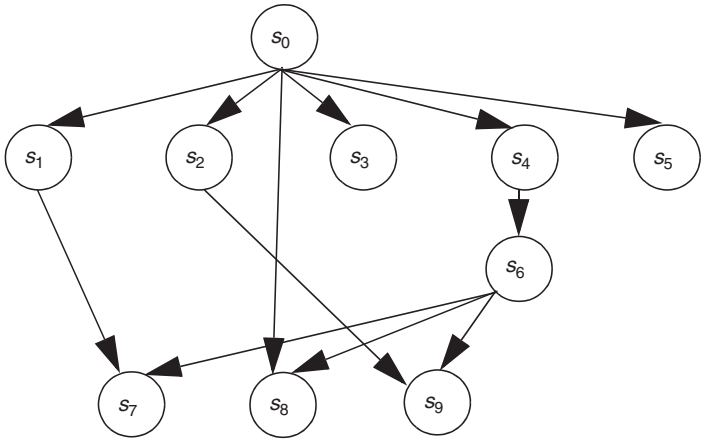
In each graph, the coefficients of the subtask execution time equation (a_i, b_i, c_i) and inter-subtask communication time equation (d_{ij} and e_{ij}) were randomly generated from uniform distributions with ranges [10–100] and [1–10], respectively. The heterogeneity factors (h_{iu}'s) of these graphs were also randomly selected from a uniform distribution with range [0.5–20].

The heterogeneous platform shown in Fig. 3 was used throughout the experiments. Below are the parameters used in the experiments, unless otherwise stated.

- Ranges of the dynamic parameters: α: [1000–5000], β: [5–25], γ: [100–500], and μ: [20–100].
- Partitioning of the dynamic parameter space: the range of each dynamic parameter is partitioned into four equal intervals (i.e., $K = 4$, see Section 4.2) and, therefore, the mapping table stores $4^4 = 256$ mappings (i.e., there are 256 regions in the four dimensional space).
- Number of randomly chosen sample scenarios within each partition (hyper-rectangle) of the dynamic parameter space: 10 (i.e., $N = 10$, see Section 4.2).
- The GA is executed 10 times for each sample scenario, from which the best mapping is chosen (thus, for a single graph, the GA was executed a total of $K^4 \times N \times 10 = 256 \times 10 \times 10 = 25,600$ times to build the mapping table).
- Estimated reconfiguration time: 1000.
- Crossover and mutation probabilities for the GA: both 0.4 (these values were chosen according to the light/moderate load results in [58]).

Two randomly generated 20-iteration execution profiles of dynamic parameters were used for each graph (see Table 2). The dynamic parameters for Profile B change eight times more rapidly, on average, than those for Profile A. In the profiles, iteration 0 is the initialization iteration. In this study, the dynamic parameter values of iteration 0 are chosen to be the mean values of the respective dynamic parameter ranges. The dynamic parameter values shown on iteration i ($i > 0$) simulate the actual dynamic parameter values *observed* after the task graph finishes iteration i execution.

Both Profiles A and B were generated randomly based on a single parameter: the mean percentage change in dynamic parameter values, called Δ. Specifically, given Δ, an increment factor, denoted by δ_{i-1}, was randomly chosen from a uniform distribution with a range [0.5Δ–1.5Δ], and its sign had a 0.5 probability of being positive. Then, a dynamic parameter for iteration i, say μ_i, was given by: $\mu_i = \mu_{i-1} \pm \delta_{i-1}\mu_{i-1}$ (the same is done for α and γ; β is given by μ/ζ where ζ was randomly chosen from a uniform distribution with a range of [4.0, 6.0]). In generating the profiles, the dynamic parameters are bounded within the specified ranges using the following method. If, say, a dynamic parameter goes beyond the upper bound (because of a positive δ_{i-1}), then the value of δ_{i-1} is negated to force the dynamic parameter to decrease rather than increase, and the reverse



(a)

s_i	a_i	b_i	c_i
s_1	9	24	49
s_2	42	61	9
s_3	42	50	43
s_4	2	34	47
s_5	9	10	38
s_8	33	59	76
s_7	63	45	29
s_8	56	14	54
s_9	48	68	10
	36	25	29

(b)

$s_i \rightarrow s_j$	d_{ij}	e_{ij}
$s_0 \rightarrow s_1$	5	3
$s_0 \rightarrow s_2$	6	6
$s_0 \rightarrow s_3$	2	1
$s_0 \rightarrow s_4$	7	5
$s_0 \rightarrow s_5$	3	7
$s_0 \rightarrow s_8$	5	6
$s_1 \rightarrow s_7$	3	2
$s_2 \rightarrow s_9$	7	7
$s_4 \rightarrow s_6$	4	2
$s_6 \rightarrow s_7$	5	6
$s_6 \rightarrow s_8$	7	8
$s_6 \rightarrow s_9$	9	10

(c)

s_i	h_{i0}	h_{i1}	h_{i2}	h_{i3}
s_1	0.4897	0.6815	0.7711	0.7503
s_2	0.2828	0.7129	0.4511	0.2725
s_3	0.2575	0.8511	0.5096	0.8779
s_4	0.6337	0.6921	0.8479	0.3451
s_5	0.8283	0.2745	0.4114	0.2836
s_8	0.7267	0.3124	0.2600	0.3354
s_7	0.3932	0.7026	0.8072	0.8066
s_8	0.5276	0.7990	0.5081	0.7942
s_9	0.5876	0.3863	0.6515	0.6472
	0.8760	0.8794	0.6965	0.2407

(d)

Fig. 6. (a) A ten-node randomly generated task graph; (b) coefficients of the subtask execution time equations; (c) coefficients of the inter-subtask communication data equations; (d) heterogeneity factors h_{iu} for the subtask.

is done when the dynamic parameter goes below the lower bound. Thus, the extent of change in dynamic parameter values could be controlled by specifying Δ: a higher value of Δ implies a more swiftly changing execution profile, which, in turn, can model a series of swiftly changing scenes in an input stream of images. The values of Δ used for Profiles A and B are 5% and 40%,respectively.

5.3. An illustrative example

To examine the performance of the On–Off approach, first consider the results of scheduling a 10-node random task graph using the two execution profiles. The structure and parameters of the 10-node random task graph are shown in Fig. 6. Detailed results of using the four approaches for Profile A are shown in Table 3. Below are the definitions of the data columns.

- $t(map[i-1])$: this is the task execution time of iteration i using the mapping *chosen* at the end of iteration $i-1$, denoted by map$[i-1]$. Here, it should be noted that at the end of iteration $i-1$, a new mapping will be determined but such a mapping would not be used for iteration i if the reconfiguration time offsets the gain of remapping. Thus, a mapping chosen at the end of iteration $i-1$ could be a new mapping or the same mapping used for iteration $i-1$. In the case of On–Off, the new mapping considered is the representative mapping for the region containing the dynamic parameter values at iteration $i-1$ stored in the mapping table. In the case of ECT, the new mapping considered is the one determined using the ECT

Table 3
Results for the ten-node random graph using Profile A ("total" below is the total task execution time for 20 iterations)

i	On–Off			ECT			GA On-line	Ideal
	$t(\text{map}[i-1])$	$t(\text{tab}[i-1])$	rc	$t(\text{map}[i-1])$	$t(\text{ect}[i-1])$	rc	$t(\text{ga}[i-1])$	$t(\text{ga}[i])$
0	—	54,391	1000	—	74,516	1000	—	46,228
1	47,508	46,045	1000	76,384	76,029	0	47,508	47,508
2	47,993	47,286	0	79,227	75,866	1000	49,095	45,280
3	48,301	51,679	0	75,375	74,586	0	50,306	45,856
4	48,921	51,679	0	76,912	74,755	1000	48,520	43,611
5	44,035	51,679	0	74,966	71,491	1000	53,579	40,937
6	49,121	51,679	0	76,226	75,020	1000	48,410	46,496
7	53,225	52,891	0	79,468	77,336	1000	51,161	47,580
8	55,303	52,891	1000	81,944	79,878	1000	55,447	52,809
9	56,355	51,679	1000	83,053	82,556	0	52,846	50,539
10	63,288	63,958	0	86,042	87,381	0	55,887	53,733
11	56,631	52,610	1000	82,874	88,226	0	58,984	51,973
12	58,025	55,142	1000	87,122	81,860	1000	63,183	53,733
13	57,410	63,958	0	87,644	87,182	0	57,948	54,311
14	59,774	53,073	1000	85,449	87,898	0	56,267	52,328
15	58,932	60,754	0	86,152	87,311	0	59,472	55,476
16	55,878	58,723	0	81,710	81,883	0	55,878	55,878
17	58,200	59,774	0	81,434	76,982	1000	56,981	51,227
18	60,966	63,958	0	88,524	81,581	1000	56,961	54,483
19	63,071	63,958	0	91,701	84,474	1000	60,043	59,846
20	65,608	—	—	95,498	93,014	—	58,528	53,842
Total	1,115,545			1,688,705			1,097,004	1,065,040

algorithm with the exact dynamic parameters at iteration $i-1$.

- $t(tab[i-1])$: this is the task execution time of the representative mapping stored in the off-line mapping table, denoted by tab$[i-1]$, for the region containing the dynamic parameter values at iteration $i-1$ stored in the mapping table. This is the value stored in the mapping table, instead of the task execution time by applying tab$[i-1]$ to the dynamic parameters at iteration $i-1$. The mapping tab$[i-1]$ may or may not be chosen for iteration i.
- *rc*: the reconfiguration cost, if remapping is performed.
- $t(ect[i-1])$: this is the execution time of the mapping, denoted by ect$[i-1]$, determined using the ECT algorithm with the dynamic parameters at iteration $i-1$. The mapping may or may not be chosen for iteration i.
- $t(ga[i-1])$: this is the task execution time of iteration i by applying the mapping determined by the GA using the dynamic parameter values from iteration $i-1$. This mapping is applied whether the remapping is justified by the gain or not. The mapping found at iteration $i-1$ (by the GA using the dynamic parameter values from iteration $i-1$) is incorporated in the initial population of the GA at iteration i. This GA on-line method is included for comparison only; applying the GA on-line for dynamic scheduling is infeasible due to the long execution time required by the GA. The GA on-line approach may require more reconfigurations than the On–Off method because it searches for a customized mapping at each iteration. Thus, to conservatively compare these approaches, the reconfiguration time for GA on-line is not considered.
- $t(ga[i])$: this is the task execution time of iteration i determined by the GA using the exact dynamic parameter values from iteration i. This is, therefore, the ideal case, and is impossible in practice because the actual values of the dynamic parameters for iteration i cannot be known before the execution of iteration i begins. Furthermore, the solutions found by both the On–Off (tab$[i-1]$) and the GA on-line (ga$[i-1]$) approaches are incorporated into the initial population of the GA. This is done to determine the "best" possible solution as a reference for comparison. For reasons similar to those given for GA on-line, reconfiguration time for GA Ideal is not considered.

As can be seen from Table 3, the On–Off approach of dynamically using off-line-derived mappings generated much smaller total execution time $(1,115,545)$ compared to that of using the ECT algorithm $(1,668,705)$. The improvement is approximately 33%. The On–Off approach consistently resulted in performance that was comparable to the infeasible GA on-line scheme (about 2% worse) and was only marginally outperformed by the Ideal (but impossible) method (about 5% worse). Recall that the strategy underlying the On–Off approach is to provide a mechanism by which mappings derived from time-consuming static heuristics, such as the GA, can be employed on-line; i.e., using the On–Off approach allows the power of the GA heuristic to be

achieved on-line. Indeed, one very interesting observation is that at some iterations (i.e., iterations 2, 3, 5, 8, 11, 12, 13, and 15), the On–Off approach generated shorter mapping execution times than the GA on-line (e.g., see Table 4 for the substantially different mappings used by the four approaches at iteration 5). This can be explained by the fact that the GA on-line mapping is optimized for the $i-1$ iterations dynamic parameters and thus, this mapping may not be very suitable for the ith iteration if the parameters vary considerably. The On–Off approach, however, is more robust to rapidly changing dynamic parameters because its mapping is based on good average performance over a wide range of dynamic parameters.

The results of scheduling the ten-node random graphs for Profile B are shown in Table 5. As expected, more reconfigurations were performed for this profile than for Profile A because the dynamic parameter values change much faster. Once again the ECT approach resulted in the worst performance with a total execution time that was approximately 45% longer than that of the On–Off approach. The On–Off approach was only slightly outperformed by the infeasible GA on-line method (about 5%) and about 20% worse than the impossible Ideal method. It should be noted that the higher degradation is partially due to the higher reconfiguration overhead of the On–Off method, which is not included in the GA on-line or Ideal execution times. Similar to the case of Profile A, at some iterations (i.e., iterations 1, 2, 8, 9, 12, and 17) the On–Off approach produced shorter task execution times than those of the infeasible GA on-line scheme.

5.4. Effectiveness of the sampling strategy

Some additional experiments using larger graphs were conducted to further test the effectiveness of the sampling strategy used. Specifically, ten 50-node random graphs were used and the following different approaches to generate representative mappings were compared:

- the mid-point of the hyper-rectangle is chosen as the only sample scenario for generating the representative mapping (called Scheme 1),
- a randomly selected point within the hyper-rectangle is chosen as the only sample scenario (called Scheme 2);
- ten sample scenarios in the hyper-rectangle are examined but for each sample scenario the GA is executed without incorporating the solution generated by the ECT algorithm as one of the members in the initial population (and the mappings of the ten sample scenarios are applied to all other sample scenarios in the hyper-rectangle, with the mapping giving the best average performance selected) (called Scheme 3);
- the approach used throughout all previous experiments—like Scheme 3 except the GA is executed with the ECT solution as a seed chromosomes (called Scheme 4).

Table 4
Mappings for iteration 5 of Profile A used by the (a) On–Off approach (adopted after iteration 1), (b) ECT approach (adopted after iteration 4), (c) GA on-line approach, and (d) Ideal approach

s_i	Proc type	ρ	$t_u(s_i)$	Comm.	C_{uv}
(a) *On–Off approach*					
s_0	1	10	11,217	$s_0 \to s_1$	77
s_1	1	13	10,864	$s_0 \to s_2$	184
s_2	2	10	13,367	$s_0 \to s_3$	34
s_3	2	5	12,199	$s_0 \to s_4$	2551
s_4	3	6	4190	$s_0 \to s_5$	212
s_5	2	2	18,553	$s_0 \to s_8$	1523
s_6	0	9	12,175	$s_1 \to s_7$	64
s_7	2	8	18,346	$s_2 \to s_9$	3048
s_8	0	15	9355	$s_4 \to s_6$	1562
s_9	3	10	4738	$s_6 \to s_7$	2281
				$s_6 \to s_8$	223
				$s_6 \to s_9$	7600
(b) *ECT approach*					
s_0	0	10	8060	$s_0 \to s_1$	2321
s_1	3	4	9906	$s_0 \to s_2$	1526
s_2	1	16	18,558	$s_0 \to s_3$	792
s_3	3	16	4928	$s_0 \to s_4$	3839
s_4	3	4	4825	$s_0 \to s_5$	194
s_5	0	14	21,664	$s_0 \to s_8$	2281
s_6	0	2	41,444	$s_1 \to s_7$	1551
s_7	0	16	13,440	$s_2 \to s_9$	176
s_8	2	12	11,798	$s_4 \to s_6$	1562
s_9	1	14	14,655	$s_6 \to s_7$	167
				$s_6 \to s_8$	3040
				$s_6 \to s_9$	2534
(c) *GA on-line approach*					
s_0	1	10	11,217	$s_0 \to s_1$	95
s_1	2	10	8090	$s_0 \to s_2$	1526
s_2	0	7	8100	$s_0 \to s_3$	26
s_3	1	2	10,836	$s_0 \to s_4$	2551
s_4	3	10	3691	$s_0 \to s_5$	1764
s_5	0	10	23,510	$s_0 \to s_8$	1523
s_6	2	8	27,098	$s_1 \to s_7$	64
s_7	1	11	24,203	$s_2 \to s_9$	5334
s_8	0	15	9355	$s_4 \to s_6$	891
s_9	3	16	3787	$s_6 \to s_7$	183
				$s_6 \to s_8$	3040
				$s_6 \to s_9$	4343
(d) *Ideal approach*					
s_0	0	10	8060	$s_0 \to s_1$	85
s_1	0	5	8517	$s_0 \to s_2$	2287
s_2	2	11	12,811	$s_0 \to s_3$	29
s_3	0	1	11,600	$s_0 \to s_4$	3839
s_4	3	14	3484	$s_0 \to s_5$	1764
s_5	1	4	14,568	$s_0 \to s_8$	1523
s_6	1	16	15,449	$s_1 \to s_7$	57
s_7	0	10	16,799	$s_2 \to s_9$	3048
s_8	1	8	9183	$s_4 \to s_6$	1033
s_9	3	14	4011	$s_6 \to s_7$	1523
				$s_6 \to s_8$	201
				$s_6 \to s_9$	5058

Note that Scheme 1 is equivalent to using a uniform distribution for representative values of dynamic parameters suggested in [10] (recall that the only other approach in [10] forced the application developer to guess what representative values to use). The above four approaches were applied to ten 50-node random graphs using Profile A and the

Table 5
Results for the ten-node random graph using Profile B ("total" below is the task execution time for 20 iterations)

i	On–Off			ECT			GA On-line	Ideal
	$t(\text{map}[i-1])$	$t(\text{tab}[i-1])$	rc	$t(\text{map}[i-1])$	$t(\text{ect}[i-1])$	rc	$t(\text{ga}[i-1])$	$t(\text{ga}[i])$
0	—	54,391	1000	—	76,029	1000	—	47,980
1	61,988	69,217	0	107,754	103,630	1000	66,150	61,988
2	36,411	38,707	0	63,998	61,852	1000	45,692	34,972
3	49,142	45,126	1000	72,559	79,625	0	44,712	40,318
4	43,516	26,908	1000	51,686	51,648	0	38,337	28,418
5	38,740	34,726	1000	64,508	49,440	1000	35,719	32,825
6	50,402	48,248	1000	74,503	74,365	0	44,250	29,231
7	34,390	50,402	0	50,107	49,296	0	29,842	27,753
8	41,947	33,684	1000	62,863	45,772	1000	48,713	31,601
9	45,940	48,911	0	83,088	85,888	0	48,095	44,184
10	57,662	48,178	1000	75,503	76,404	0	55,606	42,139
11	64,180	62,401	1000	111,327	112,449	0	69,058	61,897
12	39,930	42,374	0	67,829	68,907	0	40,476	39,930
13	46,949	40,904	1000	59,054	47,534	1000	46,350	31,203
14	46,677	45,483	1000	84,529	67,120	1000	45,041	41,910
15	58,622	41,498	1000	54,991	56,205	0	34,473	26,400
16	61,211	55,142	1000	78,358	80,091	0	57,254	51,032
17	46,193	42,374	1000	69,973	72,405	0	46,243	36,274
18	56,042	41,947	1000	85,659	72,229	1000	55,192	44,846
19	49,145	38,707	1000	52,644	57,459	0	47,570	31,678
20	51,900	—	—	68,733	63,337	—	48,931	43,881
Total	995,987			1,447,666			947,704	830,460

total mapping execution times were noted. The averages of these execution times were determined and the results were normalized with respect to those of Scheme 4. The results obtained are as follows: 1.23 (Scheme 1), 1.19 (Scheme 2), 1.16 (Scheme 3), and 1.00 (Scheme 4). These results lead to the conclusion that the relationship between the parameters space and the mappings space is highly irregular and, as such, more random sample scenarios are needed to more accurately "characterize" a good representative mapping for a hyper-rectangle. Finally, as expected, the solutions of the GA without using mappings determined by ECT are worse. Given these findings, Scheme 4 was used throughout all other experiments.

5.5. Results for larger graphs with different structures

To investigate the effects of graph sizes and structures on the performance of the On–Off approach, more extensive experiments were done for larger task graphs (recall that for each type of graphs, the test set contains ten graphs for each size of 10, 50, 100, and 200 nodes). Fig. 7 shows the average normalized total execution times of the ECT, On–Off, and GA on-line approaches with respect to the Ideal method. The normalized total execution time of each test case is calculated by dividing the total execution time of a particular approach (e.g., ECT) by that of the Ideal method. Each point on the curves gives the average value of ten test cases. Recall that the GA on-line approach is infeasible, the Ideal approach is impossible, and neither include reconfiguration time in their resultant execution time (the ECT and On–Off methods, however, do include reconfiguration time).

The execution time of the infeasible GA on-line was at most 20% greater than the Ideal approach (which assumed advance knowledge of the iteration i parameter values). The execution time of the On–Off was at most 30% greater than the *infeasible* GA on-line. Moreover, comparing the results of Profile A with that of Profile B, in most cases there was no significant increase in relative performance degradation for increasing incremental changes (Δ) in dynamic parameter values.

However, the ECT approach performed much worse, particularly for large random graphs. An explanation for this phenomenon is that because the ECT algorithm employs a strictly greedy scheduling method, the effect of making mistakes at early stages of scheduling can propagate until the whole graph is completely scheduled. The adverse impact of such a greedy approach can be more profound for larger graphs. The ECT approach did not perform as poorly for the tree and fork-join graphs because these graphs tend to contain long chains of subtasks that the ECT algorithm tends to handle well [57]. Fig. 8 shows the 95% confidence intervals (i.e., given the calculated mean over the 40 test cases, the probability that the true mean is in the interval shown is 0.95; see [12]) corresponding to the results shown in Fig. 7

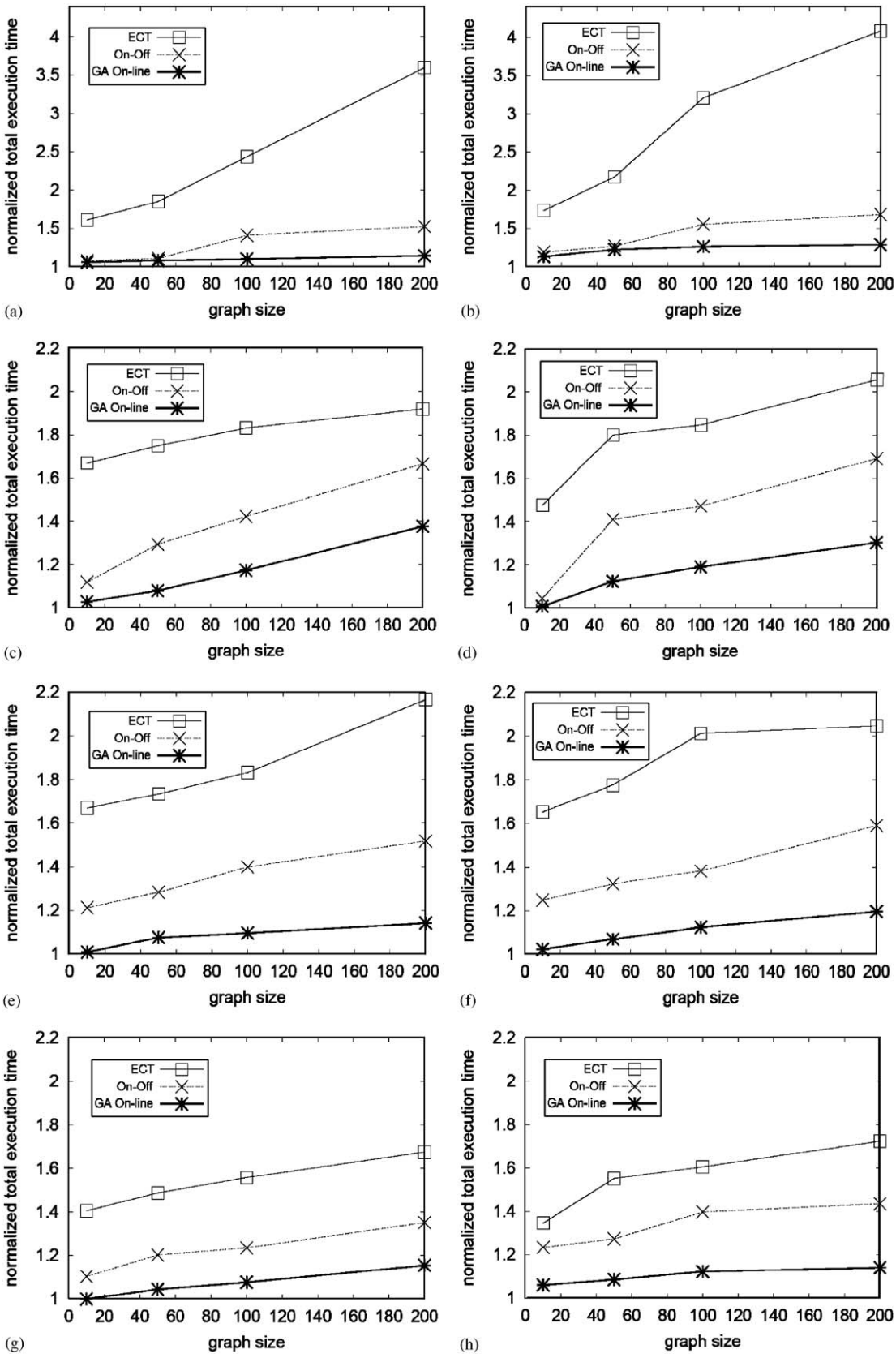


Fig. 7. Average normalized total execution times for the four types of task graphs: (a) random graphs, Profile A, (b) random graphs, Profile B, (c) in-tree graphs, Profile A, (d) in-tree graphs, Profile B, (e) out-tree graphs, Profile A, (f) out-tree graphs, Profile B, (g) fork-join graphs, Profile A and (h) fork-join graphs, Profile B (note that the scale for graphs (a) and (b) is twice that of the other graphs.

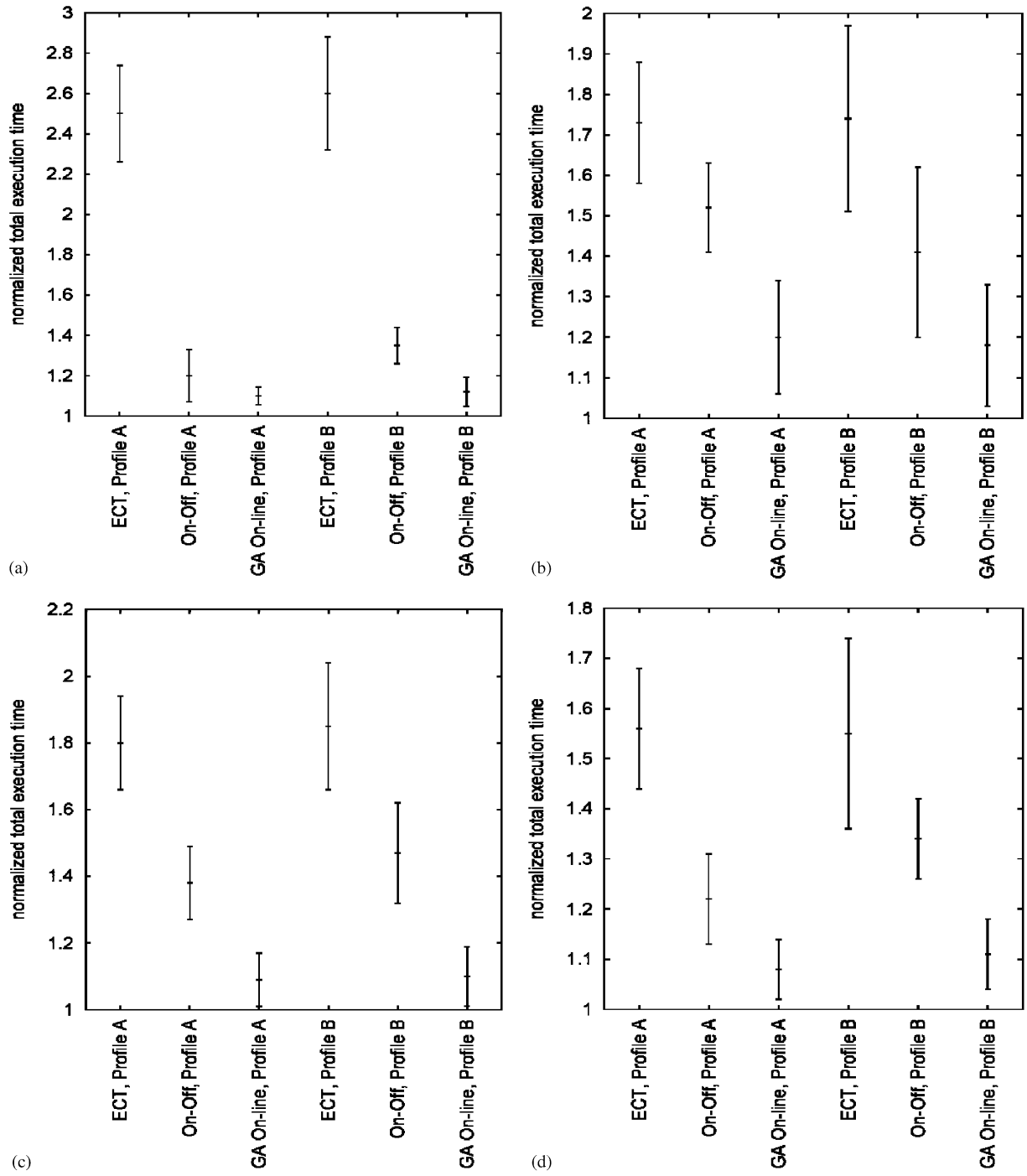


Fig. 8. The 95% confidence intervals of the normalized total execution times produced by the three approaches for all the four types of task graphs: (a) random graphs, (b) in-tree graphs, (c) out-tree graphs and (d) fork-join graphs.

and the absolute total execution times of the Ideal method are shown in Fig. 9.

5.6. Effect of the reconfiguration cost

An experiment was also conducted to explore the effect of increasing the reconfiguration cost on the performance of the On–Off approach. Ten 100-node random task graphs and Profile B (with 20 iterations) were used. The reconfiguration cost was varied as: 1000, 5000, 25,000, 50,000, and 100,000, which approximately correspond to 0.2%, 1%, 5%, 10%, and 20% of the execution time of a single iteration. The average total mapping execution times are shown in Table 6. As can be seen, the total execution times of the ECT and On–Off approaches increase moderately despite the fact that the reconfiguration cost varied over a wide range. A scrutiny

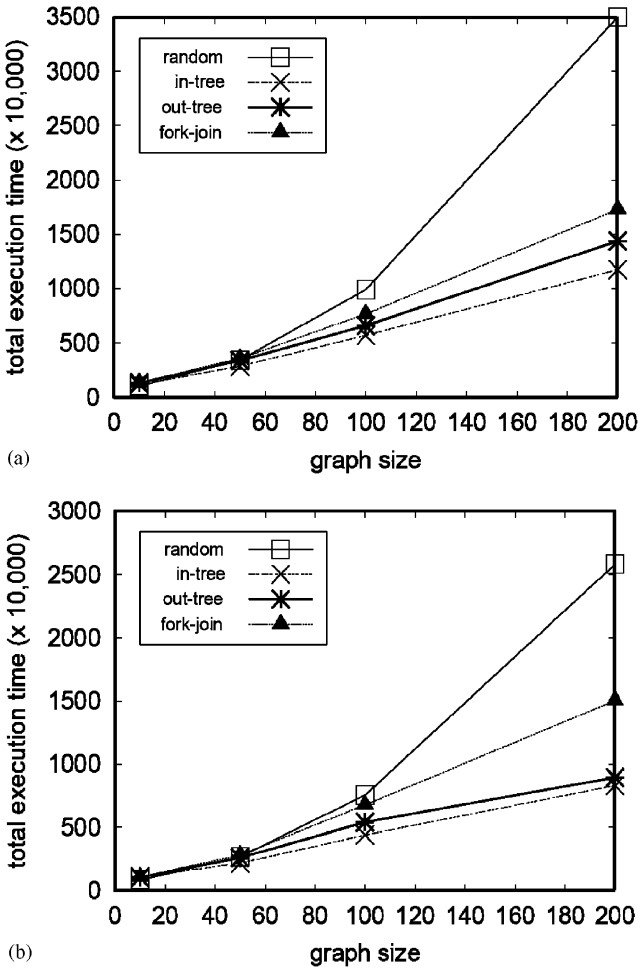


Fig. 9. Average total execution times of the Ideal approach for the four types of task graphs: (a) Profile A and (b) Profile B.

of the detailed execution traces revealed that for the ECT and On–Off approaches, a higher reconfiguration cost simply reduces the number of switches to a new mapping from one iteration to another. If reconfiguration cost is small, more remappings are performed but the aggregate reconfiguration costs do not considerably affect the total time. It appears that increasing the reconfiguration cost does not critically affect the total execution time. Most importantly, even though the total execution times of On–Off increase as expected, the On–Off approach still performs very well with respect to the other approaches.

5.7. Comparison with using a single mapping

It is interesting to compare the performance of all approaches considered thus far with a *static* approach that works by find the "best" mapping in the On–Off table. Specifically, each mapping from the 256 regions is applied to all of the 256 dynamic parameter vectors. The mapping that gives the shortest average execution time (across the 256 scenarios) is selected as the static mapping and is used in all the iterations in an execution profile. In this experiment, a suite of forty 200-node graphs was used: for each of the graph structures (random, in-tree, out-tree, and fork-join), ten graphs were used. The mapping approaches were applied to the task graphs using ten different 200-iteration execution profiles with average percentage change Δ varied from 5% to 50% in increments of 5%. These results are shown in Fig. 10. As can be seen, the relative performance of the On–Off, GA on-line, and ECT approaches is quite consistent. Furthermore, the On–Off method outperformed the static approach considerably. On the other hand, it is interesting to see that the static approach gave better performance than the ECT algorithm for test cases with small changes in dynamic parameter values, indicating that the solution quality of the GA is indeed much better than the ECT heuristic. This is due to the fact that, for this experiment, all execution profiles started in the middle of the dynamic parameter space for which the best average mapping was quite good. However, as Δ increased, the dynamic parameter values strayed further from the middle and the static approach generated progressively worse solutions (although still not as bad as a typical static mapping). For large values of Δ (e.g., larger than 25%), even the ECT approach outperformed the static approach. This illustrates that using the same mapping is not desirable for scheduling a task graph with vigorously changing attributes. In contrast, the On–Off approach was much less significantly affected by quickly varying dynamic parameters, only increasing from 1.213 to 1.297 while Δ increased from 5% to 50%.

5.8. Running time of the mapping algorithms

A limitation of the On–Off approach is the relatively long running time needed to build the mapping table. For example, while the ECT algorithm took 0.5 s to generate a mapping for a 100-node graph, the GA used in the On–Off approach required about 10 min off-line for each mapping table entry. However, because the mapping table is to be built off-line and the target heterogeneous task graph is to be used as a production job, some extra time is affordable. In addition, the efficiency of the On–Off approach can be further improved because it is possible to parallelize the GA used.

6. Conclusions

A distinctive feature of the semi-static strategy is that it approaches the off-line mapping quality of a genetic algorithm (GA) with on-line efficiency. To make the semi-static mapping methodology complete, a new technique was designed for selecting off-line mappings through partitioning and sampling the dynamic parameter space of the heterogeneous application. Experimental results indicate that with the proposed method for determining the representative static mappings, the semi-static approach is effective in that it consistently outperformed a fast dynamic mapping heuristic,

Table 6
The total execution times (average over ten graphs for each reconfiguration cost) for 100-node random task graphs using (a) Profile A, and (b) Profile B (20 iterations). The number of reconfigurations of ECT and On–Off are shown in the parentheses

Reconfig. cost	ECT	On–Off	GA on-line	Ideal
(a) *Profile A*				
1000	25,749,859	10,581,274	9,219,447	8,025,182
	(13)	(10)	—	—
5000	26,108,420	10,809,310	9,219,447	8,025,182
	(11)	(8)	—	—
25,000	26,590,023	11,093,256	9,219,447	8,025,182
	(8)	(6)	—	—
50,000	27,092,528	11,321,890	9,219,447	8,025,182
	(7)	(6)	—	—
100,000	27,502,753	11,740,731	9,219,447	8,025,182
	(7)	(6)	—	—
(a) *Profile B*				
1000	27,259,122	11,124,981	10,954,377	9,014,579
	(14)	(12)	—	—
5000	27,850,353	11,370,890	10,954,377	9,014,579
	(13)	(10)	—	—
25,000	28,601,246	11,691,931	10,954,377	9,014,579
	(11)	(8)	—	—
50,000	29,381,251	12,017,445	10,954,377	9,014,579
	(9)	(7)	—	—
100,000	29,991,021	12,529,694	10,954,377	9,014,579
	(9)	(7)	—	—

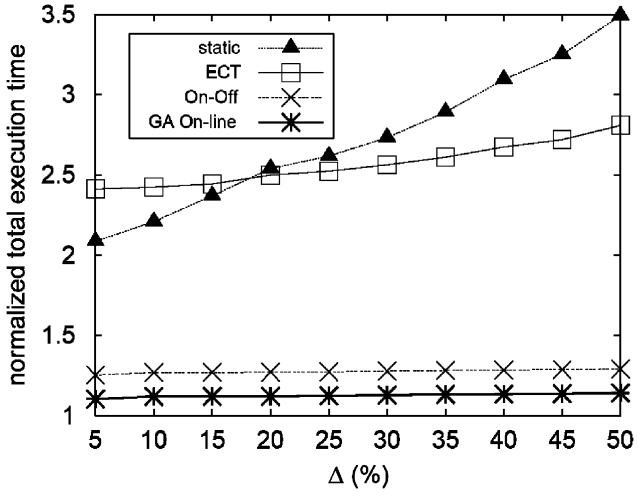


Fig. 10. Average total execution times (normalized by the execution times of the Ideal approach) for a suite of forty 200-node task graphs using various values of Δ (200-iteration profiles were used).

and yields reasonable performance compared with the infeasible approach of directly using the GA on-line for a wide range of task graph structures. Indeed, the sampling strategy performs even better than using the GA on-line (based on parameter values of the previous iteration) in several instances. Furthermore, it is found that the performance of the methodology is effective even for handling swiftly changing dynamic parameters in these heterogeneous applications. In addition, the semi-static approach outperformed the best static method by a considerable margin. A limitation of such a semi-static approach is the additional off-line execution time needed to build the mapping table. However, because the mapping table is built off-line and the target heterogeneous task graph is used as a production job, some extra time is affordable.

Acknowledgments

The authors thank the anonymous reviewers (in particular, Reviewer 1) for the constructive comments that have helped to improve this paper. Thanks are also due to Muthucumaru Maheswaran and Tracy D. Braun for their helpful suggestions. A preliminary version of portions of this paper appeared in the Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms, and Networks.

References

[1] I. Ahmad, A. Ghafoor, Semi-distributed load balancing for massively parallel multicomputer systems, IEEE Trans. Software Eng. 17 (10) (October 1991) 987–1004.

[2] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, Characterizing resource allocation heuristics for heterogeneous computing systems, in: A.R. Hurson (Ed.), Advances in Computers, Parallel, Distributed, and Pervasive Computing, vol. 63, Elsevier, Amsterdam, The Netherlands, 2005, pp. 91–128.

[3] H.M. Alnuweiri, V.K. Prasanna, Parallel architectures and algorithms for image component labeling, IEEE Trans. Pattern Anal. Mach. Intell. 14 (10) (October 1992) 1014–1034.

[4] P. Baglietto, M. Maresca, M. Migliardi, N. Zingirian, Image processing on high-performance RISC systems, Proc. IEEE 84 (7) (July 1996) 917–930.
[5] R. Bajaj, D.P. Agrawal, Improving scheduling of tasks in a heterogeneous environment, IEEE Trans. Parallel Distributed Systems 15 (2) (February 2004) 107–118.
[6] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, Scheduling strategies for master–slave tasking on heterogeneous processor platforms, IEEE Trans. Parallel Distributed Systems 15 (4) (April 2004) 319–300.
[7] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, R.F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distributed Comput. 61 (6) (June 2001) 810–837.
[8] L.L. Bölöni, D.C. Marinescu, Robust scheduling of metaprograms, J. Scheduling 5 (5) (September 2002) 395–412.
[9] J.R. Budenske, R.S. Ramanujan, H.J. Siegel, Modeling ATR applications for intelligent execution upon a heterogeneous computing platform, Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics, October 1997, pp. 649–656.
[10] J.R. Budenske, R.S. Ramanujam, H.J. Siegel, A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms, J. Supercomput. 12 (4) (October 1998) 387–406.
[11] E.A. Carmona, M.D. Rice, Modeling the serial and parallel fractions of a parallel program, J. Parallel Distributed Comput. 13 (3) (November 1991) 286–298.
[12] C.G. Cassandras, Discrete Event Systems: Modeling and Performance Analysis, Irwin, Homewood, IL, 1993.
[13] A. Choudhary, R. Thakur, Connected component labeling on coarse grain parallel computers: an experimental study, J. Parallel Distributed Comput. 20 (1) (January 1994) 78–83.
[14] C.H. Chu, E.J. Delp, H.J. Siegel, F.J. Weil, A.B. Whinston, A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architectures, J. Parallel Distributed Comput. 6 (3) (June 1989) 598–622.
[15] N. Copty, S. Ranka, G. Fox, R.V. Shankar, A data parallel algorithm for solving the region growing problem on the connection machine, J. Parallel Distributed Comput. 21 (1) (April 1994) 160–168.
[16] P. David, S. Balakirsky, D. Hillis, A real-time automatic target acquisition system, Proceedings of the International Conference on Unmanned Vehicle Systems, July 1990, pp.183–198.
[17] P. David, P. Emmerman, S. Ho, A scalable architecture system for automatic target recognition, Proceedings of the 13th AIAA/IEEE Digital Avionics Systems Conference, October 1994, pp. 414–420.
[18] T.H. Einstein, Mercury Computer Systems' Modular Heterogeneous RACE Multicomputer, Proceedings of the Sixth Heterogeneous Computing Workshop, April 1997, pp. 60–71.
[19] M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996.
[20] M.M. Eshaghian, J.G. Nash, M.E. Shaaban, D.B. Shu, Heterogeneous algorithms for image understanding architecture, Proceedings of the Workshop on Computer Architectures for Machine Perception, December 1993, pp. 286–292.
[21] T. Fahringer, Estimating and optimizing performance for parallel programs, Computer 28 (11) (November 1995) 47–56.
[22] D. Fernandez-Baca, Allocating modules to processors in a distributed systems, IEEE Trans. Software Eng. SE-15 (11) (November 1989) 1427–1436.
[23] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.
[24] A. Ghafoor, J. Yang, A distributed heterogeneous supercomputing management system, Computer 26 (6) (June 1993) 78–86.
[25] N. Guil, E.L. Zapata, Fast Hough transform on multiprocessors: a branch and bound approach, J. Parallel Distributed Comput. 45 (1) (August 1997) 82–89.
[26] K. Hwang, Z. Xu, M. Arakawa, Benchmark evaluation of the IBM SP2 for parallel signal processing, IEEE Trans. Parallel Distributed Systems 7 (5) (May 1996) 522–536.
[27] A.A. Khokhar, G.W. Cook, L.H. Jamieson, E.J. Delp, Coarse-grained algorithms and implementations of structural indexing-based object recognition on Intel touchstone delta, Proceedings of the 12th International Conference on Pattern Recognition, vol. 3, October 1994, pp. 279–283.
[28] A.A. Khokhar, V.K. Prasanna, M. Shaaban, C.-L. Wang, Heterogeneous computing: challenges and opportunities, Computer 26 (6) (June 1993) 18–27.
[29] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs onto multiprocessors, IEEE Trans. Parallel Distributed Systems 7 (5) (May 1996) 506–521.
[30] G.O. Ladd Jr., Practical issues in heterogeneous processing systems for military applications, Proceedings of the Sixth Heterogeneous Computing Workshop, April 1997, pp. 162–169.
[31] C. Lee, Y.-F. Wang, T. Yang, Global optimization for mapping parallel image processing tasks on distributed memory machines, J. Parallel Distributed Comput. 45 (1) (August 1997) 29–45.
[32] Q. Liu, R.J. Sclabassi, M.L. Scheuer, M. Sun, A two-step method for compression of medical monitoring video, Proceedings of the 25th Annual International IEEE Engineering in Medicine and Biology Society, vol. 1, September 2003, pp. 845–848.
[33] S. Madala, J.B. Sinclair, Performance of synchronous parallel algorithms with regular structures, IEEE Trans. Parallel Distributed Systems 2 (1) (January 1991) 105–116.
[34] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distributed Comput. 59 (2) (November 1999) 107–131.
[35] M. Maheswaran, T.D. Braun, H.J. Siegel, Heterogeneous distributed computing, in: J. Webster (Ed.), Encyclopedia of Electrical and Electronics Engineering, vol. 8, Wiley, New York, NY, 1999, pp. 679–690.
[36] D.C. Marinescu, J.R. Rice, On high level characterization of parallelism, J. Parallel Distributed Comput. 20 (1) (January 1994) 107–113.
[37] D.G. Morris III, D.K. Lowenthal, Accurate data redistribution cost estimation in software distributed shared memory systems, Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'2001), June 2001, pp. 62–71.
[38] D.M. Nicol, J.H. Saltz, Dynamic remapping of parallel computations with varying resource demands, IEEE Trans. Comput. 37 (9) (September 1988) 1073–1087.
[39] C.-I. Park, T.-Y. Choe, An optimal scheduling algorithm based on task duplication, IEEE Trans. Comput. 51 (4) (April 2002) 444–448.
[40] A. Radulescu, C. Nicolescu, A.J.C. van Gemund, P.P. Jonker, CPR: mixed task and data parallel scheduling for distributed systems, Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'2001), April 2001.
[41] S. Ramaswamy, S. Sapatnekar, P. Banerjee, A framework for exploiting task and data parallelism on distributed memory multicomputers, IEEE Trans. Parallel Distributed Systems 8 (11) (November 1997) 1098–1116.
[42] Remos, http://www-2.cs.cmu.edu/~cmcl/remulac/remos.html, 2003.
[43] U. Rencuzogullari, S. Dwarkadas, Dynamic adaptation to available resources for parallel computing in an autonomous network of workstations, Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'2001), June 2001, pp. 72–81.

[44] K.C. Sevcik, Characterizations of parallelism in applications and their use in scheduling, Performance Eval. Rev. 17 (1) (May 1989) 171–180.
[45] G. Shao, R. Wolski, F. Berman, Modeling the cost of redistribution in scheduling, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
[46] H.J. Siegel, J.K. Antonio, R.C. Metzger, M. Tan, Y.A. Li, Heterogeneous computing, in: A.Y. Zomaya (Ed.), Parallel and Distributed Computing Handbook, McGraw-Hill, New York, NY, 1996, pp. 725–761.
[47] H.J. Siegel, T.D. Braun, H.G. Dietz, M.B. Kulaczewski, M. Maheswaran, P.H. Pero, J.M. Siegel, J.E. So, M. Tan, M.D. Theys, L. Wang, The PASM Project: a study of reconfigurable parallel computing, Proceedings of the Second International Symposium on Parallel Architectures, Algorithms, and Networks, June 1996, pp. 529–536.
[48] H.J. Siegel, H.G. Dietz, J.K. Antonio, Software support for heterogeneous computing, in: A.B. Tucker Jr. (Ed.), The Computer Science and Engineering Handbook, CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.
[49] H.J. Siegel, T. Schwederski, W.G. Nation, J.B. Armstrong, L. Wang, J.T. Kuehn, R. Gupta, M.D. Allemang, D.G. Meyer, D.W. Watson, The design and prototyping of the PASM reconfigurable parallel processing system, in: A.Y. Zomaya (Ed.), Parallel Computing: Paradigms and Applications, International Thomson Computer Press, London, UK, 1996, pp. 78–114.
[50] M. Singh, S. Singh, Image segmentation optimisation for X-ray images of airline luggage, Proceedings of the 2004 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS'2004), July 2004, pp. 10–17.
[51] M. Srinivas, L.M. Patnaik, Genetic algorithms: a survey, Computer 27 (6) (June 1994) 17–26.
[52] J. Subhlok, P. Lieu, B. Lowekamp, Automatic node selection for high performance applications on networks, Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'1999), May 1999, pp. 163–172.
[53] M. Tan, H.J. Siegel, A stochastic model for heterogeneous computing and its application in data relocation scheme development, IEEE Trans. Parallel Distributed Systems 9 (11) (November 1998) 1088–1101.
[54] M. Tan, H.J. Siegel, J.K. Antonio, Y.A. Li, Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system, IEEE Trans. Parallel Distributed Systems 8 (8) (August 1997) 857–871.
[55] B. Veeravalli, W.H. Min, Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times, IEEE Trans. Parallel Distributed Systems 15 (3) (March 2004) 273–288.
[56] J.G. Verly, R.L. Delanoy, Model-based automatic target recognition (ATR) system for forwardlooking groundbased and airborne imaging Laser Radars (LADAR), Proc. IEEE 84 (2) (February 1996) 126–163.
[57] Q. Wang, K.H. Cheng, List scheduling of parallel tasks, Inform. Process. Lett. 37 (5) (March 1991) 291–297.
[58] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distributed Comput. 47 (1) (November 1997) 8–22.
[59] F.J. Weil, L.H. Jamieson, E.J. Delp, Dynamic intelligent scheduling and control of reconfigurable parallel architectures for computer vision/image processing, J. Parallel Distributed Comput. 13 (3) (November 1991) 273–285.
[60] T. Yang, C. Fu, Heuristic algorithms for scheduling iterative task computations on distributed memory machines, IEEE Trans. Parallel Distributed Systems 8 (6) (June 1997) 608–622.

Yu-Kwong Kwok is an Associate Professor in the Department of Electrical and Electronic Engineering at the University of Hong Kong (HKU). Before joining the HKU in August 1998, he was a Visiting Scholar for 1 year in the Parallel Processing Laboratory at the School of Electrical and Computer Engineering at the Purdue University. He recently served as a Visiting Associate Professor at the Department of Electrical Engineering—Systems at the University of Southern California from August 2004 to July 2005, on his sabbatical leave from HKU. He received his B.Sc. degree in Computer Engineering from the University of Hong Kong in 1991, the M.Phil. and Ph.D. degrees in Computer Science from the Hong Kong University of Science and Technology (HKUST) in 1994 and 1997, respectively. His research interests include distributed computing systems, wireless networking, and mobile computing. He is a Senior Member of the IEEE. He is also a Member of the ACM, the IEEE Computer Society, and the IEEE Communications Society. He received the Outstanding Young Researcher Award from HKU in November 2004.

Anthony A. Maciejewski received his B.SEE., M.S., and Ph.D. degrees from the Ohio State University in 1982, 1984, and 1987. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at the Purdue University, West Lafayette. He is currently the Head of the Department of Electrical and Computer Engineering at the Colorado State University. He is a Fellow of the IEEE. A complete vita is available at: www.engr.colostate.edu/∼aam.

H. J. Siegel was appointed the George T. Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at the Colorado State University (CSU) in August 2001, where he is also a Professor of Computer Science. In December 2002, he became the first Director of the University-wide CSU Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a Professor at the Purdue University. He received two B.S. degrees from MIT, and the M.A., M.S.E., and Ph.D. degrees from the Princeton University. Prof. Siegel has coauthored over 300 published papers on parallel and distributed computing and communication. He is a Fellow of the IEEE Fellow and a Fellow of the ACM. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of six international conferences, and Chair/Co-Chair of five workshops. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/∼hj.

Ishfaq Ahmad received his B.Sc. degree in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, and a M.S. degree in Computer Engineering and a Ph.D. degree in Computer Science from the Syracuse University, New York, USA, in 1987 and 1992, respectively. His recent research focus has been on developing parallel programming tools, scheduling and mapping algorithms for scalable architectures,

heterogeneous computing systems, distributed multimedia systems, video compression techniques, and grid computing. His research work in these areas is published in over 150 technical papers in refereed journals and conferences. He is currently a Full Professor of Computer Science and Engineering in the CSE Department of the University of Texas at Arlington. AT UTA, he leads IRIS (Institute for Research In Security), a multi-disciplinary research center engaged in safety and security related technologies. He is an Associate Editor of Cluster Computing, Journal of Parallel and Distributed Computing, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Concurrency, and IEEE Distributed Systems Online.

Arif Ghafoor is currently a Professor in the School of Electrical and Computer Engineering at the Purdue University, West Lafayette, IN, and is the Director of Information Infrastructure Security Research Laboratory and Distributed Multimedia Systems Laboratory. He has been actively engaged in research areas related to database security, parallel and distributed computing, and multimedia information systems. He has published numerous technical papers in these areas. He has served on the editorial boards of various journals. He is a Fellow of the IEEE. He has received the IEEE Computer Society 2000 Technical Achievement Award for his research contributions in the area of multimedia systems.