Evaluation of a Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems

YU-KWONG KWOK[1], ANTHONY A. MACIEJEWSKI[2], HOWARD JAY SIEGEL[2],
ARIF GHAFOOR[2], AND ISHFAQ AHMAD[3]

[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong

[2]School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA

[3]Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong

Email: *ykwok@eee.hku.hk, {maciejew, hj, ghafoor}@ecn.purdue.edu, iahmad@cs.ust.hk*

Abstract—To minimize the execution time of an iterative application in a heterogeneous parallel computing environment, an appropriate mapping scheme is needed for matching and scheduling the subtasks of the application onto the processors. When some of the characteristics of the application subtasks are unknown *a priori* and will change from iteration to iteration during execution-time, a *semi-static* methodology can be employed, that starts with an initial mapping but dynamically decides whether to perform a remapping between iterations of the application, by observing the effects of these *dynamic parameters* on the application's execution time. The objective of this study is to implement and evaluate such a semi-static methodology. For analyzing the effectiveness of the proposed scheme, it is compared with two extreme approaches: a completely dynamic approach using a fast mapping heuristic and an ideal approach that uses a genetic algorithm on-line but ignores the time for remapping. Experimental results indicate that the semi-static approach outperforms the dynamic approach and is reasonably close to the ideal but infeasible approach.

1 Introduction

Heterogeneous computing (HC) encompasses a great variety of situations (e.g., see [5], [12]). This study 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 subtasks. To minimize the execution time of such an iterative application on a heterogeneous parallel computing environment, an appropriate mapping scheme is needed for matching and scheduling of the subtasks onto the processors [12]. However, when some of the characteristics of the application subtasks are unknown *a priori* and will change from iteration to iteration during execution-time, it may not be feasible or desirable to use the same off-line derived mapping throughout the whole execution of the application.

An example of such a problem domain is iterative automatic target recognition (ATR) tasks [14], where a sequence of images is received from a group of sensors and various kinds of operations are required to generate an on-going scene description. In ATR, the characteristics of a subtask's input data, such as the amount of clutter and the number of objects to be identified, may change dynamically and may lead to large variations in the subtask's processing requirements.

In such situations, a semi-static methodology [1], [2] may be employed, that starts with an initial mapping but dynamically decides whether to remap the application with a mapping previously determined off-line. This is 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. Such real-time input-data dependent remapping between iterations can be performed by using an off-line determined mapping. That is, 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. This remapping process will, in general, require a certain system reconfiguration time for relocating the data and program modules.

The application to be mapped is iterative 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 2. 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 operating system has to decide whether a remapping is to be done. This framework can be applied to any task graph structure represented as a DAG.

The objective of this study 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), which was originally proposed in [2]. The implementation of the On-Off methodology entails tackling two research issues: (a) how to select representative mappings off-line for on-line use? and (b) is this approach really beneficial compared to a strictly dynamic approach?

This research was supported by the HKU CRCG, the Hong Kong Research Grants Council (under the contract number HKUST 6076/97E), and the DARPA/ITO Quorum Program (under NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and GSA contract number GS09K99BH0250).

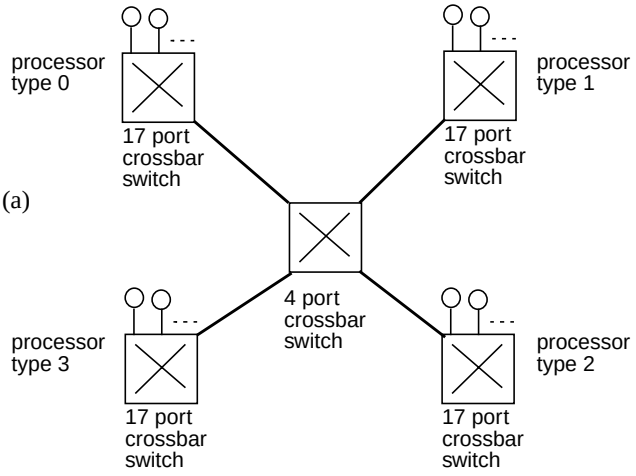
To address these issues, a novel dynamic parameter space partitioning and sampling scheme is proposed in Section 3. During the off-line phase, a genetic algorithm is used to generate high quality mappings for a range of values for the dynamic parameters. During the on-line phase, the actual dynamic parameters are observed and the off-line derived mapping table is referenced to choose the most suitable mapping. Experimental results, presented in Section 4, indicate that this semi-static approach is effective in that it consistently gave performance that was comparable to that of using the same genetic algorithm on-line with the exact dynamic parameter values for the *next* iteration (which is physically impossible). How this research differs from earlier related work, such as [8], [9], [10], [17], and further details about all aspects of this paper are in [7].

2 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 heterogeneous computing platform considered is based on the expected needs of ATR applications that are of interest to the U.S. Army Research Laboratory (e.g., [3]). Specifically, it contains four different types of processors (e.g., SHARC DSP processors and PowerPC RISC processors [4]), with 16 processors of each type (see Figure 1(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 Figure 1(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 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.

(a)



(b)

	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	0.44	7.00	7.44
type 2	10.47	5.58	4.62	0.44	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

Figure 1: (a) The target heterogeneous computing platform consisting of four types of processors with 16 in each type; (b) startup time (*S*) and transmission time per unit data (*1/R*) of the inter-processor communication channels.

An application task is modeled as a DAG, with n nodes representing subtasks s_i $(0 \le i \le n-1)$ and e edges representing inter-subtask communications. To illustrate the efficacy of 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 the form of the equations used and it is the responsibility of the application developer to use an appropriate model [1], [2].

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]). 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 [7]. 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 execution time $t_u(s_i)$ of subtask s_i on virtual machine u is modeled by the expression:

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

By differentiating this equation and equating it 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 to be transferred between two subtasks s_i and s_j consists of a fixed portion modeled by a constant d_{ij} (independent of the input) and a variable portion 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 (see Figure 1(b) for values of S_{uv} and R_{uv} based on [3], [4]). Thus, the inter-subtask communication between subtask s_i on virtual machine u and subtask s_j on virtual machine v is C_{uv} which is given by: $C_{uv}(s_i, s_j) = S_{uv} + (d_{ij} + e_{ij}\mu)/R_{uv}$.

3 The Semi-Static Mapping Approach

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 genetic algorithm 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 genetic algorithm) performs. Notice that even off-line generation of optimal mappings is infeasible because the heterogeneous mapping problem is NP-complete [6] and, thus, exponential time is needed for finding optimal solutions.

In the On-Off semi-static mapping approach, it is assumed that the ranges of the dynamic parameters are known, and are partitioned into a certain number of disjoint regions. Formally, suppose the minima and total range sizes of the dynamic parameters are given by α_{min}, β_{min}, γ_{min}, μ_{min}, ζ_α, ζ_β, ζ_γ, and ζ_μ, respectively. The parameter space $\Re$ can be partitioned into K^4 disjoint regions as follows:

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

where $\alpha_{min} + i\zeta_\alpha/K \le \alpha < \alpha_{min} + (i + 1)\zeta_\alpha/K$, and the ranges for β, γ, and μ are defined analogously. Here, the parameter space is uniformly partitioned. However, different values of K can be used for each dynamic parameter, depending upon the specifications given by the application developer.

Within each region (defined by specifying values for indices i, j, k, l), N random dynamic parameter vectors are chosen. An off-line 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 ($0 \le x \le N - 1$), denote the corresponding mapping by M_x. The mapping for each sample scenario is exhaustively evaluated for every other sample scenario in the region by applying the mapping to the DAG and computing the completion time. That is, the task execution times $t(M_x(v_y))$ for all x and y ($0 \le x, y \le N - 1$) are computed. The mapping M_x that gives the minimum average completion time $[\sum_{y=0}^{N-1} t(M_x(v_y))]/N$ 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.

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 1 in Section 4. 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, given the dynamic parameter values of iteration $i - 1$, the on-line module retrieves the mapping corresponding to the representative dynamic parameter vector *closest to* the given actual values. If the stored pre-determined execution time of the selected mapping, plus the estimated reconfiguration time, is smaller than the (simulated) actual execution time of iteration $i - 1$, a remapping is performed; otherwise, the mapping used at iteration $i - 1$ will continue to be used for iteration i.

In this study, several mapping approaches are examined. 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 fast static scheduling algorithm, called the Earliest Completion Time (ECT) algorithm, that is based on the technique presented in [15].

The method of using the ECT algorithm for scheduling the dynamic iteration tasks is as follows. 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 (with its associated estimated task execution time using the iteration $i - 1$ parameters) is then considered to be a potential new mapping for iteration i. Again, if the gain in adopting the new mapping is greater than the reconfiguration time, the new mapping will be used in iteration i.

Genetic algorithms (GAs) are a promising heuristic approach to optimization problems that are intractable. There are a great variety of approaches to GAs (see [13] for a survey). The details of our particular GA are available in [7], [16]. However, it should be noted that the On-Off approach does not rely on GAs per se and can be used with any global optimization scheme.

4 Performance Results

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 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).

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.

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 generated.

In each graph, the coefficients of the subtask execution time equation (a_i, b_i, and 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 of these graphs were also randomly selected from a uniform distribution with range 0.5 to 20.

Details of random task graph generation are given in [7].

The heterogeneous platform shown in Figure 1 was used throughout the experiments. Below are the parameters used in the experiments, unless otherwise stated.

- ranges of the dynamic parameters: α : [1,000..5,000], 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., $K = 4$) 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: ten (i.e., $N = 10$)
- the GA is executed ten 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: 1,000 (calculation of the estimated reconfiguration time is discussed in [2])
- crossover and mutation probabilities for the GA: both 0.4 (these values were chosen according to the light/moderate load results in [16])

Two randomly generated 20-iteration execution profiles of dynamic parameters were used for each graph (see Table 1). 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 Profile A and Profile B are generated randomly based on a single parameter: the mean percentage change in dynamic parameter values, called Δ. Specifically, given Δ, an increment factor, 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 (but selected to guarantee that the dynamic parameter stays within its range). Then, a dynamic parameter for iteration i, say μ_i, was given by: $\mu_i = \mu_{i-1} \pm \delta_{i-1}\mu_{i-1}$.

To examine the performance of the On-Off approach, first consider the results of scheduling a ten-node random task graph using the two execution profiles. The parameters of the ten-node random task graph are shown in Figure 2. Detailed results of using the four approaches for Profile B are shown in Table 2. Due to space limitations, results for Profile A are not shown here but can be found in [7]. Below are the definitions of the data columns.

- **t(M[i–1]):** this is the task execution time of iteration i using the mapping ***chosen*** at the end of iteration $i-1$, denoted by M[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 mapping corresponding to the scenario closest to the parameters at iteration $i-1$ in the mapping table. In the case of ECT, the new mapping considered is the one determined using the ECT algorithm with the exact dynamic parameters at iteration $i-1$.
- **t'(O[i–1]):** this is the task execution time of the mapping stored in the off-line mapping table, denoted by O[i–1], of a scenario with dynamic parameters closest to the dynamic parameters at iteration $i-1$. This is the value stored in the mapping table, instead of the task execution time by applying O[i–1] to the dynamic parameters at iteration $i-1$. The mapping O[i–1] may or may not be chosen for iteration i.
- **RC:** the reconfiguration time, if remapping is performed.
- **t(E[i–1]):** this is the execution time of the mapping determined using the ECT algorithm with the parameters at iteration $i-1$, denoted by E[i-1], which may or may not be chosen for iteration i.
- **t(G[i–1]):** this is the task execution time of iteration i by applying the mapping determined by the GA with dynamic parameters at iteration $i-1$, disregarding whether the remapping is justified by the gain or not. Also, the mapping found at iteration $i-1$ is incorporated in the initial population of the GA at iteration i. Reconfiguration time is not counted in this approach. Again, this GA On-line method is included for comparison only because applying the GA on-line for dynamic scheduling is not feasible due to the long execution time required by the GA. It should be noted that for both Ideal and GA On-line, reconfiguration time is not considered.
- **t(G[i]):** this is the task execution time of iteration i determined using the GA with the exact dynamic parameters at iteration i. This is, therefore, the ideal case which is

Table 1: Execution profiles of dynamic parameters: Profile A (average percentage change in dynamic parameter values Δ = 5%) and Profile B (Δ = 40%).

iteration	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

(a) Profile A

iteration	α	β	γ	μ
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

(b) Profile B

(a)

	coefficients			heterogeneity factors			
	a_i	b_i	c_i	h_{i0}	h_{i1}	h_{i2}	h_{i3}
s_0	9	24	49	0.4897	0.6815	0.7711	0.7503
s_1	42	61	9	0.2828	0.7129	0.4511	0.2725
s_2	42	50	43	0.2575	0.8511	0.5096	0.8779
s_3	2	34	47	0.6337	0.6921	0.8479	0.3451
s_4	9	10	38	0.8283	0.2745	0.4114	0.2836
s_5	33	59	76	0.7267	0.3124	0.2600	0.3354
s_6	63	45	29	0.3932	0.7026	0.8072	0.8066
s_7	56	14	54	0.5276	0.7990	0.5081	0.7942
s_8	48	68	10	0.5876	0.3863	0.6515	0.6472
s_9	36	25	29	0.8760	0.8794	0.6965	0.2407

(b)

	d_{ij}	e_{ij}
s_0->s_1	5	3
s_0->s_2	6	6
s_0->s_3	2	1
s_0->s_4	7	5
s_0->s_5	3	7
s_0->s_8	5	6
s_1->s_7	3	2
s_2->s_9	7	7
s_4->s_6	4	2
s_6->s_7	5	6
s_6->s_8	7	8
s_6->s_9	9	10

Figure 2: (a) Coefficients of the subtask execution time equation and heterogeneity factors h_{iu} for the subtask execution times; (b) coefficients of the inter-subtask communication data equations.

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 and the GA On-line approaches are also incorporated into the initial population of the GA. This is done in order to determine the "best" solution as a reference for comparison.

As can be seen from Table 2, the On-Off approach of dynamically using off-line derived mappings generated much smaller total execution time (995,987) compared to that of using the ECT algorithm (1,447,666). The On-Off approach consistently resulted in performance comparable to the infeasible GA On-line scheme and was only marginally outperformed by the Ideal (but impossible) method. Indeed, one very interesting observation is that at some iterations, the On-Off approach generated shorter mapping execution time than the GA On-line due to the On-Off approach being more robust to changes.

Table 2: Results for the ten-node random graph using Profile B ("total" below is the task execution time for 20 iterations).

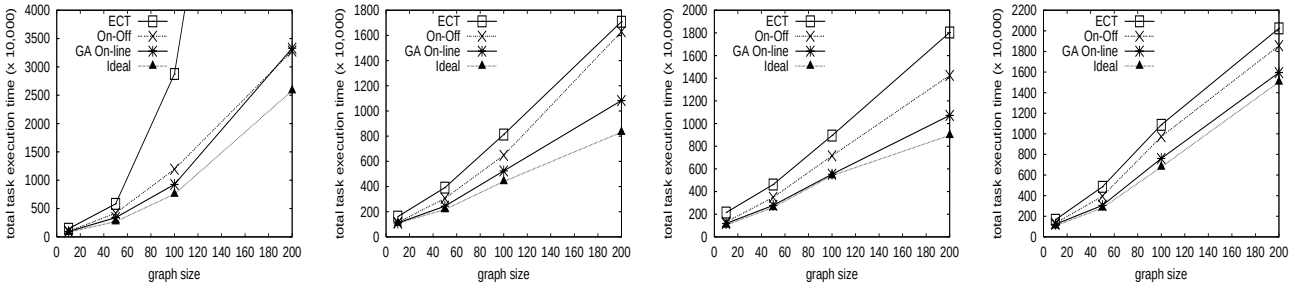
	On-Off			ECT			GA On-line	Ideal
i	t(M[i–1])	t'(O[i–1])	RC	t(M[i–1])	t(E[i–1])	RC	t(G[i–1])	t(G[i])
0	—	54391	1000	—	76029	1000	—	47980
1	61988	69217	0	107754	103630	1000	66150	61988
2	36411	38707	0	63998	61852	1000	45692	34972
3	49142	45126	1000	72559	79625	0	44712	40318
4	43516	26908	1000	51686	51648	0	38337	28418
5	38740	34726	1000	64508	49440	1000	35719	32825
6	50402	48248	1000	74503	74365	0	44250	29231
7	34390	50402	0	50107	49296	0	29842	27753
8	41947	33684	1000	62863	45772	1000	48713	31601
9	45940	48911	0	83088	85888	0	48095	44184
10	57662	48178	1000	75503	76404	0	55606	42139
11	64180	62401	1000	111327	112449	0	69058	61897
12	39930	42374	0	67829	68907	0	40476	39930
13	46949	40904	1000	59054	47534	1000	46350	31203
14	46677	45483	1000	84529	67120	1000	45041	41910
15	58622	41498	1000	54991	56205	0	34473	26400
16	61211	55142	1000	78358	80091	0	57254	51032
17	46193	42374	1000	69973	72405	0	46243	36274
18	56042	41947	1000	85659	72229	1000	55192	44846
19	49145	38707	1000	52644	57459	0	47570	31678
20	51900	—	—	68733	63337	—	48931	43881
total:	995,987			1,447,666			947,704	830,460

Some additional experiments using larger graphs were also 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 one of the seed chromosomes (called Scheme 4).

The above four approaches were applied to the ten 50-node random graphs using Profile A and the 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). Thus, using just one sample scenario is not as effective as using ten. The degradations are indeed quite significant. Furthermore, the degradation is higher if the mid-point instead of a randomly selected point within the hyper-rectangle is used. 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 subsequent experiments.

To investigate the effects of graph sizes and structures on the performance of the On-Off approach, the experiments were repeated for larger task graphs. As can be seen from Figure 3,



(a) random task graphs (b) in-tree task graphs (c) out-tree task graphs (d) fork-join task graphs

Figure 3: Total execution times for Profile B.

the total execution times of On-Off, GA On-line, and Ideal for totally random task graphs are consistently of similar values for all graph sizes. However, the ECT approach performed much worse, especially for large graphs with sizes 100 and 200 (in order to enhance the readability of the plots, the data for 200-node graphs of the ECT algorithm were excluded). 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. For other regular graphs, in most cases the performance of On-Off was only slightly inferior to the GA On-line, which in turn was slightly outperformed by the Ideal.

An experiment was also conducted to explore the effect of increasing the reconfiguration time on the performance of the On-Off approach. The total execution times of the ECT and On-Off approaches remained approximately constant despite the fact that the reconfiguration time varied over a wide range (the GA On-line and Ideal results are independent of reconfiguration time). This is due to the observation that a higher reconfiguration time simply prohibited the attempts to switch 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.

5 Conclusions

A novel strategy is presented for partitioning and sampling the dynamic parameter space of a heterogeneous application within the context of a semi-static mapping methodology. A summary of the extensive performance evaluation of this strategy is given (details in [7]). Experimental results indicate that the semi-static approach is effective in that it consistently outperformed a fast dynamic mapping heuristic, and gave reasonable performance compared with the infeasible approach of directly using the genetic algorithm on-line for a wide range of task graph structures. 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 M. Maheswaran and T.D. Braun for their helpful suggestions.

References

[1] J.R. Budenske, R.S. Ramanujan, and H.J. Siegel, "Modeling ATR Applications for Intelligent Execution upon a Heterogeneous Computing Platform," *Proc. IEEE Int'l Conf. Sys., Man, and Cyb.*, pp. 649-656, Oct. 1997.

[2] —, "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. Supercomputing*, vol. 12, no. 4, pp. 387-406, Oct. 1998.

[3] P. David, P. Emmerman, and S. Ho, "A Scalable Architecture System for Automatic Target Recognition," *Proc. 13th AIAA/ IEEE Digital Avionics Sys. Conf.*, pp. 414-420, Oct. 1994.

[4] T.H. Einstein, "Mercury Computer Systems' Modular Heterogeneous RACE Multicomputer," *Proc. 6th Heterogeneous Computing Workshop*, pp. 60-71, Apr. 1997.

[5] M.M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

[7] Y.-K. Kwok, A.A. Maciejewski, H.J. Siegel, A. Ghafoor, and I. Ahmad, "Implementation and Performance Study of a Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems," Technical Report HKUST-CS99-15, Department of Computer Science, HKUST, April 1999.

[8] C. Lee, Y.-F. Wang, and T. Yang, "Global Optimization for Mapping Parallel Image Processing Tasks on Distributed Memory Machines," *J. Parallel and Distributed Computing*, vol. 45, no. 1, pp. 29-45, Aug. 1997.

[9] D.M. Nicol and J.H. Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands," *IEEE Trans. Computers*, vol. 37, no. 9, pp. 1073-1087, Sept. 1988.

[10] S. Ramaswamy, S. Sapatnekar, and P. Banerjee, "A Framework for Exploiting Task and Data Parallelism on Distributed Memory Multicomputers," *IEEE Trans. Parallel and Distributed Sys.*, vol. 8, no. 11, pp. 1098-1116, Nov. 1997.

[11] K.C. Sevcik, "Characterizations of Parallelism in Applications and Their Use in Scheduling," *Performance Evaluation Rev.*, vol. 17, no. 1, pp. 171-180, May 1989.

[12] H.J. Siegel, J.K. Antonio, R.C. Metzger, M. Tan, and Y.A. Li, "Heterogeneous Computing," in *Parallel and Distributed Computing Handbook*, ed. A.Y. Zomaya, McGraw-Hill, New York, NY, pp. 725-761, 1996.

[13] M. Srinivas and L.M. Patnaik, "Genetic Algorithms: A Survey," *Computer*, vol. 27, no. 6, pp. 17-26, June 1994.

[14] J.G. Verly and R.L Delanoy, "Model-Based Automatic Target Recognition (ATR) System for Forwardlooking Groundbased and Airborne Imaging Laser Radars (LADAR)," *Proc. IEEE*, vol. 84, no. 2, pp. 126-163, Feb. 1996.

[15] Q. Wang and K.H. Cheng, "List Scheduling of Parallel Tasks," *Inf. Proc. Lett.*, vol. 37, no. 5, pp. 291-297, Mar. 1991.

[16] L. Wang, H.J. Siegel, V.P. Roychowdhury, and A.A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *J. Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8-22, Nov. 1997.

[17] T. Yang and C. Fu, "Heuristic Algorithms for Scheduling Iterative Task Computations on Distributed Memory Machines," *IEEE Trans. Parallel and Distributed Sys.*, vol. 8, no. 6, pp. 608-622, June 1997.