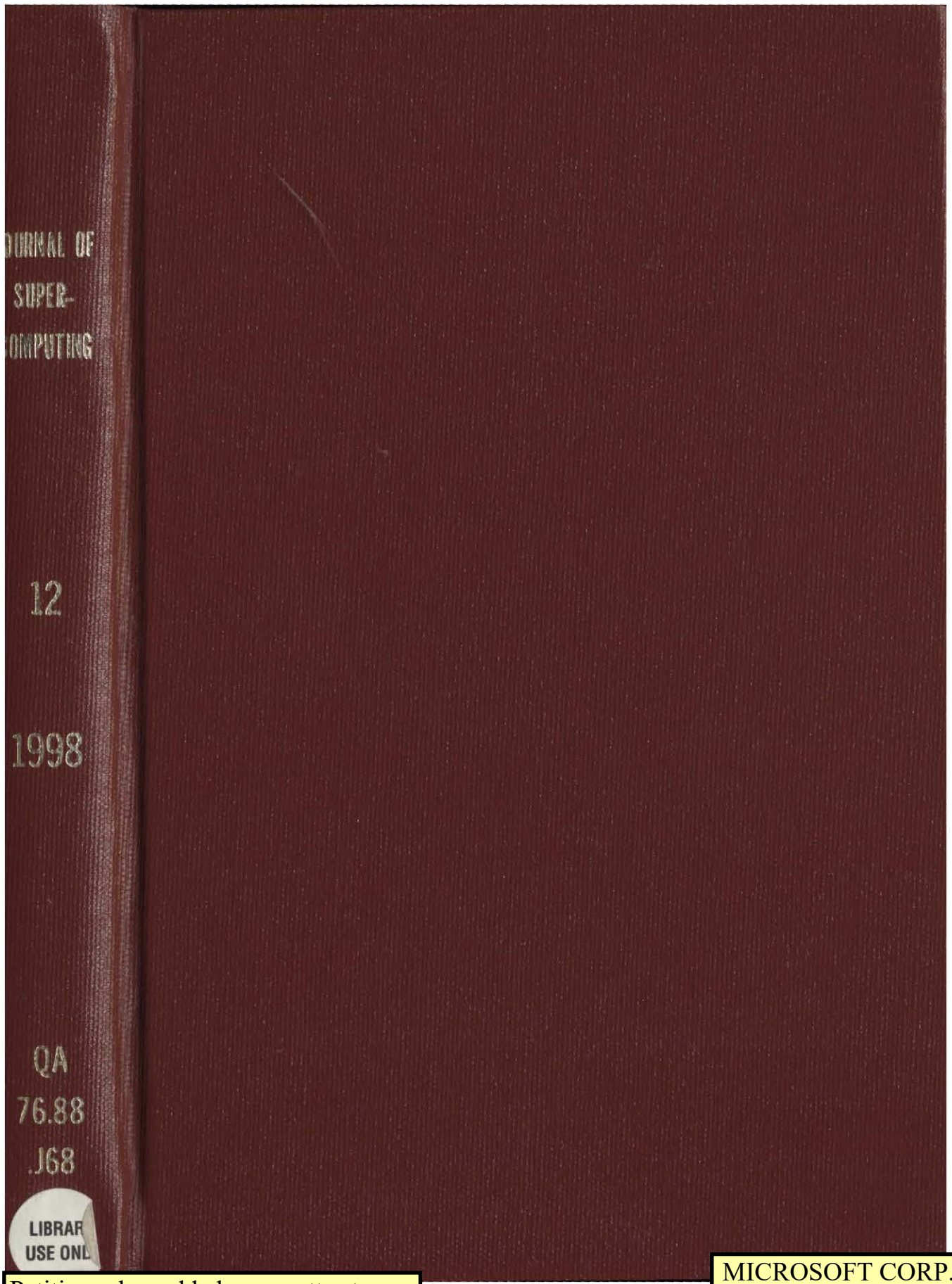

Petitioner has added romanettes to pages (i)-(vi). Otherwise, it leaves the original page numbering.

MICROSOFT CORP.
EXHIBIT 1005

A
6.88
J68

OLUME 12, NUMBER 4,
ctober 1998
SN 0920–8542

HIGH-PERFORMANCE COMPUTER DESIGN, ANALYSIS, AND USE

PORTLAND STATE UNIV
NOV 10

The Journal of Supercomputing

Kluwer Academic Publishers

PORTLAND STATE UNIVERSITY Library

THE JOURNAL OF SUPERCOMPUTING

Editor-in-Chief

Hamid Arabnia, University of Georgia

Subject Area Editors

APPLICATIONS: David H. Bailey, NASA Ames Research Center
ALGORITHMS: Kenneth Neves, Boeing Computer Services
PERFORMANCE: Joanne Martin, IBM Corporation
ARCHITECTURE AND SYSTEMS: George Michael, Lawrence Livermore National Laboratory

ISSN: 0920–8542

© 1998 Kluwer Academic Publishers, Boston/U.S.A.

No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the copyright owner.

Manufactured in the Netherlands

Board of Review

Makoto Amamiya, *Kyushu University*
Arvind, *Massachusetts Institute of Technology*
William F. Ballhaus, Jr., *Martin Marietta Astronautics*
David Barkai, *NASA Ames Research Center*
Francine Berman, *University of California at San Diego*
Alfred Brenner, *Institute for Defense Analysis*
Ernest Brickell, *CertCo*
Harvey Cragon, *The University of Texas at Austin*
Narsingh Deo, *University of Central Florida*
Jack J. Dongarra, *University of Tennessee*
Geoffrey C. Fox, *Syracuse University*
John Gregory, *Westinghouse Electric Corporation*
Milton Halem, *NASA Goddard Space Flight Center*
Harry Jordan, *University of Colorado*
Sidney Karin, *GA Technologies*
David Klappholz, *Stevens Institute of Technology*
Gordon Lyon, *National Institute of Standards and Technology*
Kenichi Miura, *Fujitsu America*
Ronald C. Mullin, *University of Waterloo*
Carl Pomerance, *The University of Georgia*
Paul B. Schneck, *MRJ Technology Solutions*
Jacob T. Schwartz, *Courant Institute of Mathematical Science*
Daniel Siewiorek, *Carnegie-Mellon University*
Burton J. Smith, *TERA Computer Company*
James E. Smith, *University of Wisconsin - Madison*
Lloyd Thorndyke, *Consultant*
Jack Worlton, *Worlton and Associates*
Stephen S.Yau, *University of Florida*

Second Class Postage Paid Rahway, N.J., Mercury Airfreight International Ltd. Inc., 2323 Randolph Avenue, Avenel, N.J. 07001
ISSN: 0920–8542 USPS: 003–163
Postmaster: Please send all address corrections to: *The Journal of Supercomputing,* c/o Mercury Airfreight International Ltd. Inc., 2323 Randolph Avenue, Avenel, N.J. 07001, USA.

VOLUME 12, NUMBER 4, OCTOBER 1998

THE JOURNAL OF SUPERCOMPUTING

CONTENTS

Distribution-Independent Hierarchical Algorithms for the N-Body Problem
Srinivas Aluru, John Gustafson, G.M. Prabhu and Fatih E. Sevilgen 303

A Robust Compile Time Method for Scheduling Task Parallelism on Distributed Memory Machines
Sekhar Darbha and Santosh Pande 325

Compiler Support for Array Distribution on NUMA Shared Memory Multiprocessors
Tarek S. Abdelrahman and Thomas N. Wong 349

An Efficient Systolic Algorithm for the Longest Common Subsequence Problem
Yen-Chun Lin and Jyh-Chian Chen 373

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
John R. Budenske, Ranga S. Ramanujan and Howard Jay Siegel 387

The Journal of Supercomputing is abstracted and/or indexed in *ISI: Current Contents: Engineering, Technology and Applied Science, ISI: Science Citation Index, ISI: Scisearch, ISI: Research Alert, ISI: Compumath Citation Index, INSPEC Information Services, Engineering Index, COMPENDEX* Plus database, Computer Literature Index, Artificial Intelligence Abstracts, Computer and Information Systems Abstracts, Information Science Abstracts, ACM Guide to Computing Literature, ACM Computing Reviews, Computer Abstracts, ISI: Automatic Subject Citation Alert, Zentralblatt fur Mathematik, Referativnyi Zhurnal, MATH (Database), CompactMATH (CD-ROM), Compuscience (Database), PASCAL Database,* and *Ei Page One.*

0920-8542(199810)12:4;1-I

CODEN JOSUED ISSN 0920-8542

PORTLAND STATE UNIVERSITY LIBRARY

The Journal of Supercomputing

Volume 12, No. 4, October 1998

Distribution-Independent Hierarchical Algorithms for the N-Body Problem ..... *Srinivas Aluru, John Gustafson, G.M. Prabhu and Fatih E. Sevilgen* 303

A Robust Compile Time Method for Scheduling Task Parallelism on Distributed Memory Machines ..... *Sekhar Darbha and Santosh Pande* 325

Compiler Support for Array Distribution on NUMA Shared Memory Multiprocessors ..... *Tarek S. Abdelrahman and Thomas N. Wong* 349

An Efficient Systolic Algorithm for the Longest Common Subsequence Problem ..... *Yen-Chun Lin and Jyh-Chian Chen* 373

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 ..... *John R. Budenske, Ranga S. Ramanujan and Howard Jay Siegel* 387

The Journal of Supercomputing is abstracted and/or indexed in ***ISI: Current Contents: Engineering, Technology and Applied Science, ISI: Science Citation Index, ISI: Scisearch, ISI: Research Alert, ISI: Compumath Citation Index, INSPEC Information Services, Engineering Index, COMPENDEX Plus database, Computer Literature Index, Artificial Intelligence Abstracts, Computer and Information Systems Abstracts, Information Science Abstracts, ACM Guide to Computing Literature, ACM Computing Reviews, Computer Abstracts, Applied Mechanics Reviews, ISI: Automatic Subject Citation-Alert, Zentralblatt für Mathematik, Referativny: Zhurnal, MATH (Database), CompactMATH (CD-ROM), Compuscience (Database), PASCAL Database, Ei Page One.***

Subscription Rates

The subscription price for Volume 12 (4 quarterly issues), including postage and handling, is:
For institutions: US$349.00 Dfl. 680,—
For individuals: US$175.00 Dfl. 320,—

Ordering Information/Sample Copies

Subscription orders and requests for sample copies should be sent to:

Kluwer Academic Publishers
Post Office Box 358
Accord Station
Hingham, MA 02018-0358, USA

or

Kluwer Academic Publishers
Post Office Box 322
3300 AH Dordrecht
THE NETHERLANDS

Subscriptions may also be sent to any subscription agent.

Printed on acid-free paper.

Photocopying. *In the U.S.A.*: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by Kluwer Academic Publishers for users registered with the Copyright Clearance Center (CCC) Transactional Reporting Service, provided that the fee of $9.50 per copy per article is paid directly to CCC. For those organizations that have been granted a photocopy license by CCC, a separate system of payment has been arranged. The fee code for users of the Transactional Reporting Service is 0920-8542/98/$9.50.

Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising purposes or promotional purposes, for creating new collective works, or for resale.

In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to Kluwer Academic Publishers, P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

ISSN 0920-8542

© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.
No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the copyright owner.

The Journal of Supercomputing, 12:387–406 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in the Netherlands

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

JOHN R. BUDENSKE AND RANGA S. RAMANUJAN
Architecture Technology Corporation, P.O. Box 24344, Minneapolis, MN 55424, USA, {budenske, ranga}@atcorp.com

HOWARD JAY SIEGEL
Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA, hj@purdue.edu

Abstract. This study focuses on a particular application domain (iterative automatic target recognition tasks) and an associated specific class of dedicated heterogeneous parallel hardware platforms. For the computational environment considered, a methodology is presented for the on-line operating system to decide heuristically whether to perform a remapping of the application onto the platform based on information generated from input data by the application during execution. If the decision is to remap, the operating system will be able to select a mapping, which is appropriate for the given state of the application, from a stored set of mappings that were previously derived with an off-line heuristic.

Keywords: Algorithm mapping, automatic target recognition, genetic algorithms, heterogeneous computing, real-time processing

1. Introduction

Heterogeneous computing (*HC*) covers a great variety of situations (e.g., see [9], [17], [22], [23], [26]). This study focuses on a particular application domain (iterative automatic target recognition (*ATR*) tasks) and an associated specific class of dedicated heterogeneous parallel hardware platforms. The contribution of this paper is that, for the computational environment considered, it presents a methodology for real-time on-line input-data dependent remapping of the application subtasks to the processors in the heterogeneous parallel hardware platform using a previously stored off-line statically determined *mapping* (i.e., a matching of application subtasks to processors and a scheduling for the execution order of these subtasks). That is, the operating system will be able to make a heuristically-based 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 previously heuristically derived and stored mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking).

This high-level operating system approach, which includes the capability for the on-line use of off-line computed mappings, is called the *IOS* (Intelligent Operating System). The IOS conceptual design, off-line components, and on-line components are depicted in Figures 1, 2, and 3, respectively. Consider the conceptual design (Figure 1). The *ATR Kernel* 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* makes a heuristically-based decision about how the partially ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. Also, the HC Kernel interacts with the *Basic Kernel* (the low-level operating system) to execute the application and monitor its execution so it can decide when to remap the subtasks 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. This design has its roots in the high-level model presented in [4] for automatic dynamic processor allocation in a partitionable parallel machine with homogeneous processors (e.g. PASM).

This paper concentrates on the operation of the HC Kernel; the other components will be discussed only to the extent needed for describing the HC Kernel (additional information on the other components is in [2], [3]). In particular, this paper focuses on (1) the

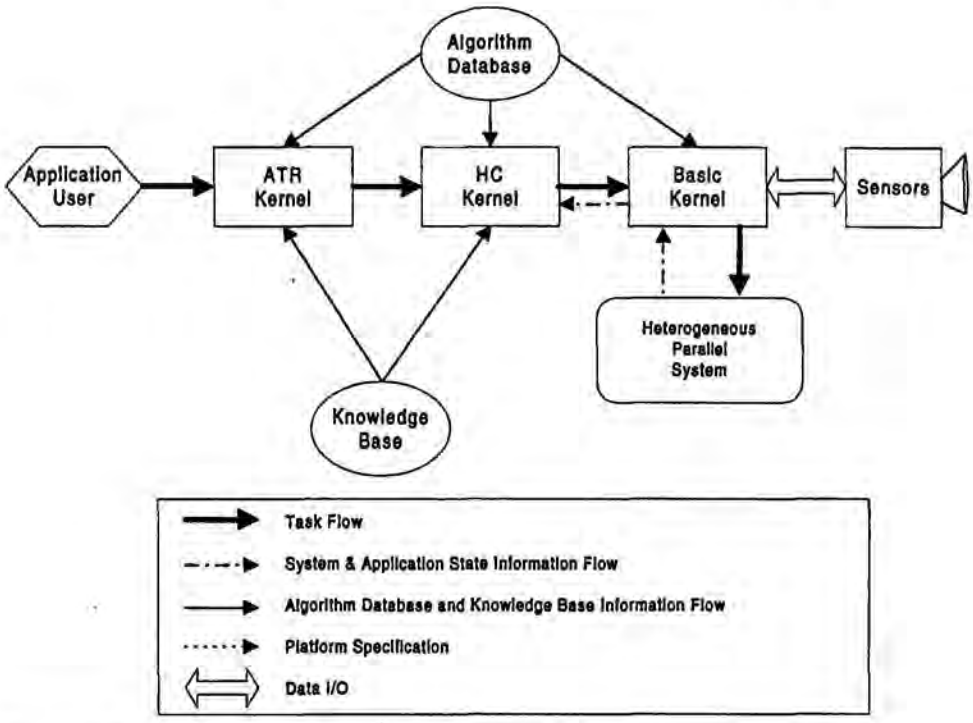

Figure 1. IOS Conceptual Design

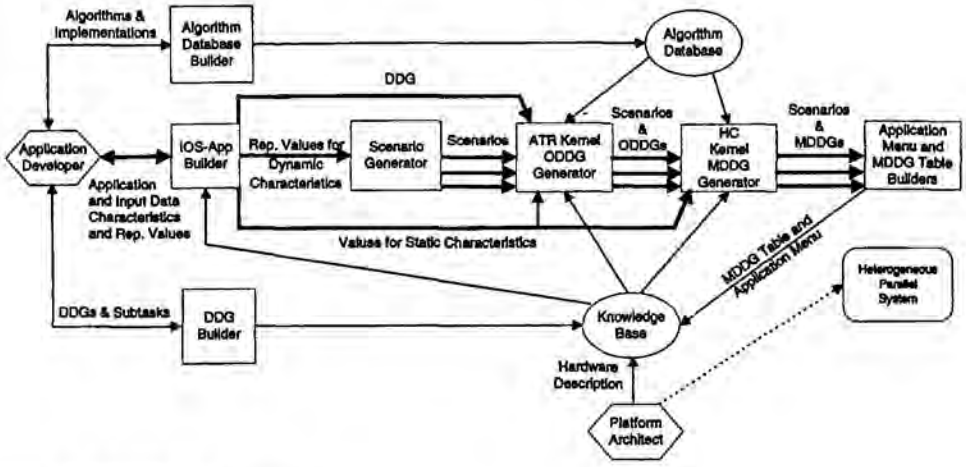

Figure 2. IOS Off-line Components

application and hardware platform characteristics that are needed to enable the use of the HC Kernel, (2) the techniques that comprise the HC Kernel, and (3) how the information needed for the HC Kernel to operate is collected. The IOS has not been implemented; such an implementation is a major undertaking and outside the scope of this paper, which is the design concepts for the HC Kernel.

The HC Kernel differs from other real-time HC mapping techniques in that it allows on-line real-time use of off-line precomputed mappings. This is significant because it is possible for off-line heuristics to produce better mappings than on-line heuristics. This is a result of the fact that off-line heuristics can have much longer execution times to search

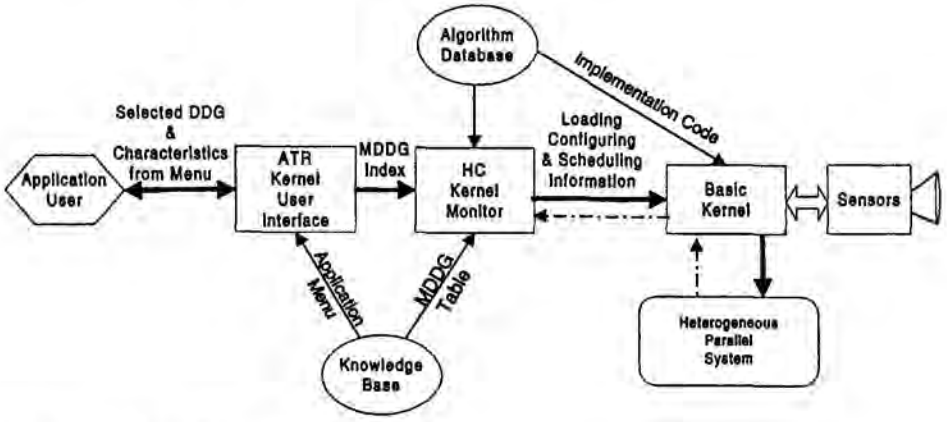

Figure 3. IOS On-line Components

for a good solution than what is practical for an on-line heuristic. Thus, with the HC Kernel, the mapping quality of an off-line time-consuming heuristic can be approached at real-time speeds.

The HC Kernel ideas can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the iterative ATR applications and platforms considered here. Examples of other such application domains are sensor-based robotics, intelligent vehicle highway systems, air traffic control, nuclear facility maintenance, weather prediction, intruder detection, and manufacturing inspection.

Thus, the goal of this paper is to introduce the ideas underlying the design and use of the HC Kernel for a particular class of application computational structure and a particular type of target hardware platform. The performance of the HC Kernel in comparison to other methods will depend upon the exact application and exact platform under consideration. The technique presented here is offered to future system designers to evaluate as an alternative for their particular situations.

The paper is organized as follows. Section 2 describes the application domain and Section 3 the heterogeneous parallel hardware platform. Overviews of the off-line and on-line components of the IOS are presented in Sections 4 and 5, respectively. More information about the HC Kernel off-line and on-line components are provided in Sections 6 and 7, respectively.

2. Application domain

Simply stated, an ATR system takes a set of images from a group of sensors and produces some type of description of the scene. The various types of processing required in an ATR system can be broadly classified into three groups: numeric computation, 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 symbolic computation (e.g., used to produce the scene description). Heterogeneous parallel architectures are ideal 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 implement 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 run time.

This work is being developed for a class of ATR applications each of which can be modeled as an iterative execution of a set of partially ordered subtasks. Each ATR application in this class is a production job that is executed repeatedly. Thus, it is worthwhile to invest off-line time in preparing an effective mapping of the application onto the

hardware platform used to execute it. The ATA (automatic target acquisition) system described in [5] and the tracking system described in [18] are examples of such iterative ATR applications.

Each application task will be an instantiation of a *DDG* (data dependency graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies among subtasks. The expected number of subtasks is 10 to 50. The DDG will be structured as a directed acyclic graph (*DAG*). The IOS is being designed for applications that will iteratively execute such a DDG. Note that while the subtasks' dependencies are represented as a DAG, subtasks themselves may contain loops.

For the initial iteration through the set of subtasks, the IOS will use information about the processing environment in its selection of algorithms for the subtasks, and their associated implementations. As part of this, the IOS will decide how to assign resources (e.g., processors) to the subtasks.

After each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image stream being processed. It is expected that the values of these parameters will change slowly. After all subtasks have completed execution for a given iteration through the DDG, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel. The HC Kernel will use the most recent values of such dynamic parameters to estimate if it is worthwhile to reconfigure the assignment of processing resources to subtasks to reduce execution time of the next iteration. If it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks. If not, the same assignment will continue to be used.

3. Heterogeneous parallel hardware platform

The type of target hardware platforms considered for this study are driven by the expected needs of the classes of ATR applications that are of interest to the U.S. Army Research Laboratory. Thus, for the intended application environment of the IOS, it is assumed that there will be up to four different types of processors, and up to a total of 64 processors (of all types combined). For example, two types may be SHARC DSP processors and PowerPC RISC processors [8]. These processors will comprise the heterogeneous parallel architecture onto which application tasks will be mapped. A system of this size should provide the real-time computing power needed for the intended application domain. The IOS approach is appropriate for larger HC platforms as well.

A small-scale example of the type of hardware platform being considered is the one described in [6]. This system was developed by the Army Research Laboratory for a real-time ATR relational template matching algorithm. Another example, although outdated, is [1], which describes a heterogeneous hardware platform designed to perform ATR research and prototyping.

All the processors of all types will have communication paths to one another. 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 to communicate is the same. For example, a Mercury daughter board can be populated with six SHARC processors that physically share a DRAM. To connect processors of different types, a VME bus can be used to provide communications among different collections of processors. In addition, the hardware platform will include (1) a workstation, for off-line IOS operations to develop an application implementation and for use as the *Application User* interface, and (2) a *Host Processor*, which will monitor the application implementation during its execution and implement the on-line HC Kernel.

For simplicity, it is assumed that if an implementation of a given subtask uses multiple 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.

The IOS design should be capable of working with any hardware platform of the type described above. A given hardware platform may be used for many different ATR applications. It is assumed that when a given ATR application is executing on a platform, that platform is dedicated to that application.

4. Overview of IOS off-line components

Figure 2 shows the off-line components of the IOS. The Knowledge Base contains a collection of DDGs. The Algorithm Database contains information about algorithms that can be used to perform the subtasks in the DDGs. The DDGs and algorithm information is supplied by the application domain expert, the *Application Developer*. The Application Developer, who is responsible for developing the application software, will typically be a different person (or people) from the Application User(s), who may know nothing about software development and may just use the ATR system as a prepackaged tool. An analogy to this in the personal computing field is the programmer who develops a software package for graphics versus the package user who draws figures with the tool without any knowledge of the details of the actual code in the software package.

To design a particular application, the Application Developer first selects a DDG. (The Application Developer can also specify a particular set of DDGs that can execute simultaneously and be treated as a single DDG [3].) Each DDG has associated with it a list of *Application Characteristic* and *Input Data Characteristic* names, whose values must be filled in when the DDG is instantiated for a given application and associated environment. (The SmartNet project uses a similar set of characteristics called "Compute Characteristics," whose values when an application is invoked are called the "Compute Characteristics Operating Point" [12].) As discussed in Section 6, the Algorithm Database will contain information about the expected execution time of each subtask algorithm implementation, and this time will be a function of the values of these characteristics (and other factors).

Each of these sets of Application and Input Data Characteristics can be divided into static and dynamic parameters. *Static parameters* are those Input Data Characteristics, such as image size, and Application Characteristics, such as type of object of interest (e.g., tank), that will not change during the execution of the application task. *Dynamic parameters*, in contrast, are those Input Data Characteristics, such as amount of clutter, and Application Characteristics, such as number of located objects to be identified, that will change during run time and can be computed by the application as it executes. When an Application Developer instantiates a DDG to implement an application in a given environment, the values for the static characteristics are known and provided by the Application Developer. For the dynamic characteristics, the Application Developer is expected to define a small set of dynamic parameters that will have the most impact on the execution time of the subtasks in the application. It is the job of the *Scenario Generator* component of the IOS to use information provided by the Application Developer to derive representative values for these dynamic parameters.

There are several ways in which the Scenario Generator can develop the representative dynamic parameter values. The sample ways to be described differs in the amount of involvement by the Application Developer, who is the domain expert who knows the most about the application and its associated subtasks. Also, recall from Section 2 that the values of these dynamic parameters are expected to change slowly, i.e., it is expected that most dynamic parameters will not continuously have dramatic changes in value from each image frame to the next.

Assume D dynamic parameters are selected for a given application. Let the number of representative choices for the i-th dynamic parameter be C_i, $0 \leq i < D$. Each set of D representative values for these D dynamic parameters, one per parameter, is called a *scenario*. The number of different scenarios that can be generated is $S = C_0 \times C_1 \times \cdots \times C_{D-1}$. The importance of picking an effective set of dynamic parameters and representative choices for parameter values is discussed later.

The best way to determine the representative choices for each dynamic parameter is to have the Application Developer specify them. The Scenario Generator can be set up to work interactively with the Application Developer to accomplish this (through the *IOS-App Builder* software component of the IOS).

Consider the following hypothetical example. The task is the tracking of vehicles on the ground, and the Application Developer has chosen four dynamic parameters: (1) number of objects of interest being tracked in the last image frame, (2) average size of the objects currently being tracked (in terms of percent of the total number of pixels in the image), (3) whether the lighting conditions in the last image frame required requantization for better contrast for display, and (4) whether there was cloud cover detected in the last image frame that required special processing. The representative choices specified for the dynamic parameters are: (1) 0, 1, 2, 4, 8, 12, (2) 5%, 10%, 15%, 20%, 40%, (3) yes or no, and (4) yes or no. Thus, $C_0 = 6$, $C_1 = 5$, $C_2 = 2$, $C_3 = 2$, and $S = 120$.

This example demonstrates another aspect of the scenario generation process: pruning. In some cases, there may be parts of the D-dimensional dynamic parameter space that have a low probability of occurring in reality. For this example, having the number of objects being tracked be eight or twelve, and having the average size of an object be 40%,

is very unlikely. The Scenario Generator will work interactively with the Application Developer (through the IOS-App Builder) to prune out such combinations of dynamic parameter values. Even if a certain improbable combination that is pruned does appear during the execution of the application, the HC Kernel will select the combination that was included that is closest to the actual run-time values of the dynamic parameters.

An alternative to having the Application Developer specify the exact representative choices for each dynamic parameter is to have the Application Developer just specify a range of values (e.g., minimum and maximum) and the associated C_i for each dynamic parameter. The Scenario Generator could then calculate a set of C_i uniformly distributed choices over the specified range. Of course, any combination of these two approaches could also be used.

Thus, the Scenario Generator constructs S scenarios, where each scenario corresponds to a particular set of representative values for the D dynamic parameters (one representative value for each dynamic parameter). The Scenario Generator passes these S scenarios to the off-line component of the ATR Kernel, as shown in Figure 2.

The job of the off-line component of the ATR Kernel is to select one or more algorithms from the Algorithm Database for each subtask of the DDG previously instantiated for the given application by the Application Developer. This is done separately for each of the S scenarios associated with the given application. The algorithm selection is based on the ability of a given algorithm to perform the image processing required by a given subtask; i.e., each subtask is assigned one or more algorithms whose *image processing capability* for that subtask and its associated Application and Input Data Characteristics is above some threshold. Thus, the selection process can use information about the application's static parameter values and that scenario's dynamic parameter values. The execution time of an implementation of the algorithm is not considered by the ATR Kernel (the HC Kernel handles this).

Performing this assignment of one or more algorithms to each subtask transforms a DDG into an *ODDG* (over-instantiated DDG), and the off-line component of the ATR Kernel is called the *ATR Kernel ODDG Generator*. Therefore, for a given application, the ATR Kernel ODDG Generator creates a distinct ODDG for each scenario.

In summary, for a given application, the Application Developer specifies a DDG, an associated static environment, and D dynamic parameters (with representative values). The Scenario Generator creates S scenarios based on the representative values of the dynamic parameters. The ATR Kernel ODDG Generator constructs S ODDGs for the given application, one for each scenario.

For each ODDG, one *MDDG* (mapped DDG) is created by the *HC Kernel MDDG Generator*. The MDDG specifies how the corresponding ODDG will be implemented and mapped onto the HC platform, as discussed further in Section 6. Therefore, S MDDGs are generated by the HC Kernel MDDG Generator for a given application and its associated static environment. These S MDDGs are sent by the HC Kernel MDDG Generator to the *MDDG Table Builder*, which constructs the *MDDG Table* for that application and its associated static environment. The MDDG Table will be indexed as a D-dimensional array, where the i-th dimension is of size C_i, i.e., an MDDG Table entry will be accessed by a list of D indices (corresponding to a scenario), where the value of the i-th index

corresponds to one of the C_i allowable representative values for the i-th dynamic parameter. MDDG Tables are stored in the Knowledge Base, as are the associated *Application Menus* constructed by the *Application Menu Builder*.

A very important question that arises is what set of dynamic parameters and representative values for these parameters should be used in the scenario generation process. The number of scenarios, S, should also be considered. The larger S is, the closer a given scenario in the MDDG Table may match a given set of actual dynamic parameter values calculated during execution of the application. It is expected that the closer this match is, the better the mapping specified by the corresponding MDDG Table entry will be. However, the larger S is, the larger the MDDG Table will be, because it will hold S MDDGs. Also, increasing S increases the off-line execution time for the ATR Kernel ODDG Generator and HC Kernel MDDG Generator; however, for the intended production ATR application domain, the off-line preparation time is much less significant than the resulting on-line execution time of the application task. The Application Developer will need to trade off these factors when selecting dynamic parameters and representative choices for these parameters, based on a knowledge of the problem domain and the sensitivity of the expected execution times of the subtasks to the values of these parameters. Thus, these decisions will depend on the given application, static parameter values, and ranges for dynamic parameter values for a given environment. In general, this may require the Application Developer to perform some experiments with sample data sets to determine the best dynamic parameters and representative values to use in order to achieve the desired system performance (i.e., use more off-line time to examine multiple sets of scenarios to determine which one will give acceptable on-line application performance using an acceptable size MDDG Table). The IOS can be designed to readily support such testing.

5. Overview of IOS on-line components

Figure 3 shows the on-line components of the IOS. Once in the field, the Application User interacts with the *ATR Kernel User Interface*, including the Application Menus. The Application Menus will be used on-line by the Application User to select a particular initial MDDG to use to invoke an application (with an associated set of Application User specified values for static parameters and initial choices of allowable representative dynamic parameters). That MDDG is passed to the *HC Kernel Monitor* to use as the initial mapping. Then, as the application is executing, the HC Kernel Monitor monitors the run-time values of the dynamic parameters at the end of each iteration through the underlying DDG to make a heuristically-based decision whether to continue with the current mapping, or to select and instantiate a new mapping (for the next iteration) from among the entries of the relevant MDDG Table (which were determined off-line). Thus, the off-line processing provides a set of predetermined mappings that the on-line processing can index in real time.

6. HC Kernel MDDG Generator

The HC Kernel MDDG Generator is the component of the off-line IOS (Figure 2) that is responsible for mapping each ODDG onto the heterogeneous parallel hardware platform creating a corresponding MDDG. Three items of information about the hardware platform will be needed by the HC Kernel MDDG Generator: (P1) algorithm implementation execution time, (P2) number of each type of processor available, and (P3) inter-processor communication time. The HC Kernel MDDG Generator uses this information when it applies the heuristic for determining an effective assignment of subtask implementations to processors.

The Algorithm Database will include one or more implementations of each algorithm (e.g., one for each processor type). The Algorithm Database must also contain the expected execution time of each algorithm implementation, typically specified as a function of type and number of processors assigned, interprocessor communication time, and certain static and dynamic Input Data and Application Characteristics. This is an expected time, rather than a definite time, because it may vary depending on the actual values of the input data being processed. This expected execution time information must be provided by the Application Developer, who also supplies the code for each implementation of a given algorithm. It is expected that algorithm implementations will be written using the number of processors (of a given type) as an input parameter whenever possible. The assumption of the availability of expected implementation execution time for each type of processor (or set of processors of the same type) is typically made for the current state of the art in HC systems (e.g., [11], [13], [20], [24]). The Application Developer can determine this information empirically. The HC Kernel MDDG Generator needs to use this information (in conjunction with the other information below) to determine the expected total application task execution time.

The second item needed is the total number and type of the processors in the platform. The HC Kernel MDDG Generator needs this to know how many processors of each type it has available to assign. This is specified by the *Platform Architect*, who is responsible for the hardware design of the system. Typically, this person (or people) is distinct from the Application Developer(s) and Application User(s), although it is assumed that the Platform Architect(s) will consult with the Application Developer(s). Referring to the earlier analogy with personal computing, the people responsible for designing the system hardware configuration will typically not be the same people who develop the graphics package.

The last item needed is a *communication matrix* indicating the time it takes for each processor in the platform to communicate with every other processor in the platform. This is also specified by the Platform Architect. Entry (i, j) in this matrix is the information needed to calculate the conflict-free time for processor i to send data to processor j. The communication time for a given pair of processors typically will have two components: a fixed latency time for the first byte to arrive, and a variable time that depends on the length of the message being transmitted (that is based on the bandwidth of the communication path). This type of matrix is used by other researchers in HC (e.g., [13], [20], [24]). This matrix is needed for the HC Kernel MDDG Generator to determine the expected inter-

subtask communication times for possible mappings of subtasks onto the platform. Recall from Section 3 that it is assumed that the target hardware platform is dedicated to this application, so the communication channels can be scheduled to prevent conflicting requests.

Consider the structure of an MDDG. Each MDDG is isomorphic to an ODDG (i.e., for each node in the ODDG, there is a unique corresponding node in the MDDG). Each MDDG node stores node–specific information that includes: (S1) which implementation (stored in the Algorithm Database) will be used for one of the algorithms in the corresponding ODDG node; (S2) pointers to the needed object code for that implementation; (S3) any additional information needed for loading that implementation onto the heterogeneous parallel system; (S4) any needed inter-subtask (i.e., inter-node) communications between that given MDDG node and any other node in the MDDG (which will also enforce data-dependency constraints among the subtasks); and (S5) the specific set of processors that will be used to execute that MDDG node. In addition to this node specific information, global MDDG information is stored with each MDDG, including: (G1) the schedule for the execution order of the MDDG nodes and inter-subtask data transfers; and (G2) the expected execution time for one iteration through the MDDG for the scenario (values of dynamic parameters) that was specified along with (and is the basis of) the corresponding ODDG. The execution time information in (G2) above is used by the HC Kernel Monitor when deciding whether to change the mapping (i.e., reconfigure), as discussed in the next section. As discussed earlier, the HC Kernel MDDG Generator passes this MDDG to the MDDG Table Builder, to be stored as part of the MDDG Table for the given application and static environment characteristics.

The Application Developer can also specify an estimate of the average overhead time to reconfigure the mapping of the given application (and associated static environment) on the hardware platform. This estimate will represent the time needed to remap the application during execution as a result of changes to the values of the dynamic parameters. The estimated average reconfiguration overhead time will be sent from the Application Developer to the HC Kernel MDDG Generator through the IOS-App Builder software. If the Application Developer does not know how to estimate this value, the Application Developer can provide the IOS-App Builder with a set of scenarios that are expected to occur frequently based on experience with the application, and the IOS-App Builder can use these to actually perform remappings among these scenarios to calculate an overhead estimate. Alternatively, the IOS-App Builder can generate a relatively small random subset of scenarios to use to calculate the estimate. Rather than derive a single estimated average overhead value, the reconfiguration time could be calculated for each of the S^2 possible old and new configuration pairs, and stored in the Knowledge Base; however, this would require an excessive amount of space to store and an excessive amount of off-line time to calculate, and, thus, it is not advisable.

The HC Kernel MDDG Generator will pass the average reconfiguration overhead time estimate to the MDDG Table Builder to be stored as part of the MDDG Table for this application. This overhead time will be used by the HC Kernel Monitor when deciding whether or not to perform a reconfiguration.

Thus, the HC Kernel MDDG Generator gets from the ATR Kernel ODDG Generator an ODDG and an associated scenario, it creates an MDDG containing the information specified above, and then passes this MDDG to the MDDG Table Builder that constructs a complete table and stores it in the Knowledge Base. The rest of this section will examine how the HC Kernel MDDG Generator can derive the information that comprises the MDDG.

In the HC field, the MDDG node specific items (S1), (S4), and (S5) defined earlier in this section are part of the process of matching subtasks in a task graph to processors in the heterogeneous parallel system. The MDDG node specific items (S2) and (S3) above are adaptations of standard operating system functions that will need to be implemented, but will not be discussed further here. The global MDDG item (G1) is referred to as the scheduling component of a mapping in an HC environment.

For general HC, deriving an optimal matching and scheduling is intractable (i.e., it is known to be an NP-complete problem that requires exponential execution time to perform an exhaustive search of the space of possible solutions [10]). This is true even when all execution and communication times can be determined statically (i.e., they are not input-data dependent). Consider the intended application domain and hardware platforms. As an example, assume there are four types of processors, with 16 processors of each type, and assume the application consists of 50 subtasks. Then, an exhaustive search of all possible assignments of subtasks to machines will take time proportional to $(4 \times 16)^{50}$. Thus, a heuristic is used, as is common in the heterogeneous computing field [22], [23].

The structure of the HC Kernel MDDG Generator is such that any good heuristic could be employed. The heuristic that is used could be one that can be executed during run time, such as a levelized-min-time heuristic (e.g., [16]), or an off-line heuristic, such as a genetic algorithm (e.g., [20], [24], [28]). In general, an off-line heuristic can find a better mapping than an on-line run-time heuristic because its execution time can be orders of magnitude longer than that of the run-time algorithm. An example of this difference in quality of matchings is provided in [28]. However, because of the longer execution times of genetic-algorithm-based heuristics, it would not be appropriate to execute a genetic algorithm while a real-time application is running in order to decide how to reconfigure resources based on the actual values of the dynamic parameters at the end of a given iteration through the underlying DDG.

For this application domain, it is possible to use off-line precomputed mappings to reconfigure resources in real time. In particular, the IOS will: (I1) allow the HC Kernel MDDG Generator to use genetic algorithms (or other off-line heuristics) to determine a matching and scheduling (i.e., MDDG) off-line for each scenario associated with a production ATR application task; (I2) allow the Application User to select an initial MDDG when the application's execution is initiated (through the ATR Kernel User Interface); and (I3) allow the HC Kernel Monitor to select a new MDDG during execution if desired based on the actual values of the dynamic parameters at the end of an iteration through the corresponding DDG. Item (I1) above is the subject of this section, item (I2) is part of the ATR Kernel User Interface (not the HC Kernel), and item (I3) will be covered in the next section.

While the HC Kernel MDDG Generator can incorporate any appropriate off-line mapping heuristic, to describe the design ideas involved in the HC Kernel it will be assumed that a genetic algorithm is used. *Genetic algorithms* are a promising heuristic approach to optimization problems that are intractable [7], [14], [15]. There are a great variety of approaches to genetic algorithms; many are surveyed in [19], [25].

In this environment, the genetic algorithm is a guided heuristic search through the space of possible matchings and schedulings (called *solutions*). The genetic algorithm in [28] was found to be very successful, and used an HC model that is quite compatible with the situation here. However, in [28] each subtask was assigned a single machine, so, for the type of platforms described in Section 3, the "chromosome" representing the matching will have to be adapted to allow for multiple processors of the same type to be assigned to a subtask. A technique for doing this has been developed [2], but its discussion is outside the scope of this paper.

The approach in [28] differs from other genetic algorithm approaches to matching and scheduling for heterogeneous systems in the literature ([20], [24], [27]) in many ways. The most significant difference from [20] and [24] is that the [28] model of a heterogeneous system is more realistic (e.g., [24] assumes an unlimited number of machines). The main difference between [28] and [27] is that in [28] it is assumed that there is a given target hardware platform, whereas [27] selects processors to be included in the platform and uses processor cost as a co-metric. Thus, for the HC Kernel for the application and platform environments discussed in Sections 2 and 3, it is most appropriate to build on the [28] genetic approach.

For the intended application domain here, the genetic algorithm needs the following information to create a matching and scheduling, i.e., to transform an ODDG into an MDDG: (M1) the structure of the underlying DDG; (M2) the expected execution time of each subtask on a set of processors (of the same type) assigned, as a function of the type of the processors and the number of processors; (M3) the inter-subtask data transfers needed, in terms of formats and expected sizes of the data items to be transferred; (M4) the expected time to send data from one processor to another as a function of the size of the data item to be transferred; and (M5) the number of each type of processor that is in the hardware platform. The genetic algorithm selects a subset of possible solutions and then evaluates them using the information in items (M1) to (M5) above. The genetic algorithm uses the results of the evaluations of these possible solutions to generate a new set of possible solutions. This process iterates until one of the stopping criteria is met (e.g., no improvement in solution quality after a given fixed number of iterations within the genetic algorithm). When the genetic algorithm stops iterating, the best solution found is used as the mapping for the MDDG.

How the HC Kernel MDDG Generator gets each of the above information items is now considered. Much of the relevant information flow is depicted in Figure 2.

The HC Kernel MDDG Generator gets the item (M1) from the ODDG that is passed to it from the ATR Kernel ODDG Generator. The genetic algorithm uses a topological sort (i.e., a valid total ordering) of the subtasks in the ODDG to establish the order in which it evaluates the nodes of the ODDG.

Item (M2) is provided by the Application Developer and stored in the Algorithm Database. Information about static and dynamic parameters may be needed in some cases (e.g., the size of subtask input data blocks). The HC Kernel MDDG Generator will receive the static information from the IOS-App Builder and the dynamic (scenario) information from the ATR ODDG Generator (see Figure 2). The expected execution times for subtasks calculated from this information are used by the genetic algorithm in its evaluation of possible solutions.

Recall that each subtask of the underlying DDG corresponds to a node of the ODDG that may contain more than one possible algorithm to perform the subtask. For each of these algorithms, there is at least one implementation that can execute on the hardware platform, and possibly more than one. When the genetic algorithm evaluates a given possible mapping solution (i.e., assignment of resources), it selects the implementation for the subtask that has the smallest expected execution time for the resources assigned to that subtask. If no implementation is available for that assignment, that mapping is considered invalid by the genetic algorithm and is discarded from the set of possible mappings.

There may be cases when there are inter-subtask implementation interactions that must be considered when selecting the implementation for a given subtask. One example of how this can occur is when some subtask *A* sends a data block, e.g., an image, to a subtask *B*, and the image is stored using different formats for the two subtasks' implementations selected (e.g., assigning rectangular subimages to processors in a multiprocessor implementation for subtask *A* and row striping of subimages in a multiprocessor implementation for subtask *B*). When this occurs, overall implementations based on each data format are considered, and the one with the smallest execution time is used as a basis for selecting the implementations for subtasks *A* and *B* for this possible solution mapping. For the intended application domain, this should not cause a significant time penalty relative to the total execution time for the genetic algorithm to generate a mapping for the MDDG being constructed.

An alternate approach for handling mismatched data block storage organizations is to allow each subtask implementation to use its own choice of organization and convert between organizations during run time. For the intended real-time ATR applications, this option will not be considered; however, if it becomes desirable to consider this option in the future, the genetic algorithm framework of [28] will allow it to be included.

Item (M3) is provided by the Application Developer and stored in the Algorithm Database as part of the input/output parameter descriptions (see [2], [3]). This inter-subtask data transfer information is used by the genetic algorithm (in conjunction with the information in item (M4)) in its evaluation of possible solutions.

Item (M4) is provided by the Platform Architect and stored in the Knowledge Base as part of the hardware description. Together with the information from item (M3), the genetic algorithm can evaluate the expected inter-subtask communication times for a given possible mapping solution.

Item (M5) is provided by the Platform Architect and stored in the Knowledge Base as part of the hardware description. It is used by the genetic algorithm to know the upper bound on the number of each type of processor that can be assigned. It should be noted that in some cases the optimal mapping (in terms of total execution time for an iteration

through the ODDG) may not use all available processors. That is, in some cases, the overhead involved in using all of the processors may result in a longer total execution time for an iteration than if only a subset of the available processors are used, as discussed in [21].

Thus, using all of this information, the genetic algorithm can create a matching and scheduling to be included in the MDDG for the given ODDG. As part of the evaluation for the potential solutions, the genetic algorithm computes the expected total execution time per iteration for each solution. For the mapping solution chosen, this expected total time is stored with the MDDG, as mentioned earlier. The completed MDDG is sent to the MDDG Table Builder.

The HC Kernel can also be used to provide some fault tolerance for the overall system. To accomplish this, the Platform Architect first specifies the fault-free system as the target platform. The application is processed using the IOS methodology presented here, forming an MDDG Table for that application and target platform. After this is done, the Platform Architect can specify as the target platform a faulty variation of the fault-free system, e.g., with one fewer functioning type "Y" processor. The application is processed again, this time using the faulty variation of the system as the target platform, forming a second MDDG Table, which corresponds to the same application but using the faulty variation of the system as the target platform. By doing this, the HC Kernel MDDG Generator will construct an MDDG Table of mappings that can be accessed and used in real time should a hardware fault of the type planned for occur. For the example above, if a type "Y" processor becomes faulty, the HC Kernel will switch from using the fault-free case MDDG Table and use the MDDG Table corresponding to this single-fault case instead.

This can be done for different faulty variations of the hardware system. Each MDDG Table will correspond to a given platform variation for that application with its associated static environment. Obviously, each variation considered increases the off-line computation and the storage that will be needed. The trade-off of this extra off-line time to prepare these MDDG Tables and the storage to keep them against the importance of handling faults for a given operational situation must be considered.

Given the symmetric assumption of Section 3, if an MDDG Table is prepared for the case of a faulty type "Y" processor, that MDDG Table can handle any one of the type "Y" processors becoming faulty. Thus, if one faulty variation on the hardware system is generated for each type of processor (e.g., four, as discussed in Section 3), then the case of any one processor of any type becoming faulty can be handled.

7. HC Kernel Monitor

The HC Kernel Monitor is the on-line component of the IOS (Figure 3) responsible for (1) establishing the initial mapping of the given application onto the hardware platform, and (2) monitoring the execution of the application and, at the end of each iteration through the corresponding DDG, deciding if and how the mapping of the application onto the hardware platform should be changed based on information about the actual values of the

dynamic parameters. To establish the initial mapping, the HC Kernel Monitor uses the MDDG index (scenario) and associated MDDG Table identifier passed to it from the ATR Kernel User Interface. With this index and identifier, the HC Kernel Monitor can access from the appropriate MDDG Table in the Knowledge Base the MDDG entry selected by the Application User (see Figure 3). The HC Kernel Monitor then passes the relevant information to the Basic Kernel (i.e., loading, configuring, and scheduling information). The Basic Kernel accesses the implementation code directly from the Algorithm Database using pointers provided with the loading information. The Basic Kernel can then begin execution of the application task.

During execution of the application, the HC Kernel Monitor receives updated actual values of the dynamic parameters at the end of each iteration through the corresponding DDG. Specifically, after all subtasks' implementations have completed execution for a given iteration of the DDG corresponding to the application, and before the next iteration begins, the latest values of these dynamic parameters will be sent to the HC Kernel Monitor from the Basic Kernel. The HC Kernel Monitor will use the most recent values of these dynamic parameters to estimate if changing the matching and scheduling will reduce the expected execution time of the next iteration through the corresponding DDG. Thus, this decision is made in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration.

If it is desirable to change the mapping, then it is the responsibility of the HC Kernel Monitor to select a new matching and scheduling (i.e., a new MDDG entry from the given MDDG Table) to use for the next execution iteration through the corresponding DDG. If not, the same mapping will continue to be used.

To determine if the current mapping should be changed, the HC Kernel Monitor performs the following sequence of steps.

(1) For each dynamic parameter, the representative value included in the allowable choices for that parameter (as specified by the Scenario Generator) is found that is closest in absolute difference to the current actual value for that parameter. This is done for all D dynamic parameters for this application. The resulting vector of D representative values is used as the approximation of the set of current actual dynamic parameters in terms of a precomputed scenario (MDDG Table index). Call this new MDDG Table index A. If MDDG Table index A is the same MDDG Table index as the one used for the current iteration, the next iteration of the application proceeds with the same mapping as the current iteration. If MDDG Table index A is not the same as the one used for the current iteration, then steps (2) through (5) are performed.
(2) The HC Kernel Monitor then accesses the MDDG Table for this application, using A as the index. The expected execution time for an iteration through the MDDG Table entry corresponding to MDDG Table index A is used as an estimate of what the expected execution time will be for the actual dynamic parameters for the next iteration of the application if the remapping occurs. Call this time $T1$.
(3) The actual execution time of the last (current) iteration through the application's corresponding DDG is used as an estimate for the time to execute the next iteration

with the current actual dynamic parameters with the current mapping (this is only an estimate because some or all of these parameter values may have been different at the beginning of the current iteration and changed sometime during the execution of this iteration). Call the actual execution time for the current iteration *T2*.

(4) When making the decision to reconfigure the resources, in addition to comparing *T1* and *T2*, the HC Kernel Monitor must also consider the time required to perform the reconfiguration (e.g., any code and data movements that the reconfiguration will require). The average reconfiguration overhead time estimate stored with the MDDG Table is used. Call this overhead time *TO*. (The HC Kernel Monitor could start with the estimate provided with the MDDG Table and then, for each time a reconfiguration is performed for this application, measure the actual overhead time and use this experiential information to modify the most recent estimate in some weighted way.)

(5) If $(T1 + TO) < T2$, then the HC Kernel Monitor instructs the Basic Kernel to change the mapping to the one corresponding to the MDDG Table entry for MDDG Table index *A*.

Thus, the HC Kernel Monitor can use the actual values of dynamic parameters at the end of each execution iteration of the application to make remapping decisions and select new mappings derived by a time-consuming off-line heuristic. As can be observed, the execution of the computationally simple steps (1) to (5) above can be done in real time, causing relatively negligible overhead compared to the expected execution time of an iteration through an ATR DDG.

The decisions made by the HC Kernel Monitor are based on heuristics and approximations. Thus, pathological cases could cause a bad decision to be made. However, pathological worst cases will also exist for fixed mappings (based on average performance) and for on-line schedulers. In general, the ideas underlying the HC Kernel Monitor will lead to reduced overall application execution time for the environment under consideration. Obviously, the HC Kernel methodology should be used only in those situations where it proves effective compared to other approaches, which can be determined only by analysis and experimentation for the exact application and target hardware platform under consideration.

As an example of a possible pathological case that could occur, if $T1 = T2 - i$ and $TO = i + 1$, then $(T1 + TO) > T2$, and reconfiguration would not be done. If the values of the dynamic parameters do not change for the next 20 iterations, the execution time for those iterations will total approximately $20 \times T2$. If reconfiguration had been done, the execution time for those iterations would total approximately $20 \times T1 = 20 \times T2 - 20 \times i$. Because the values of dynamic parameters depend on the input data, the number of iterations that will occur with no changes to the dynamic parameter values can never be predicted with certainty. If situations such as the above do appear to occur frequently for a given application, it would be possible to instrument the HC Kernel Monitor to collect the relevant statistics and then develop and add special rules. For this example, a new rule may be that if after so many iterations with no change to the dynamic parameter values, if the relationship between *TO* and $(T2 - T1)$ is smaller than some threshold, perform the remapping. However, there is no guarantee that the dynamic parameter values will not

change during the next iteration, so the decision to include such rules must be made carefully.

Thus, by studying properties of the application domain, the IOS builder may decide to fine tune the HC Kernel Monitor in different ways. The HC Kernel design has the flexibility to allow such tuning.

8. Summary

This study focused on a design for an HC Kernel for iterative ATR tasks and an associated specific class of dedicated heterogeneous parallel hardware platforms. For the computational environment considered, an HC Kernel was presented for making real-time on-line input-data dependent remappings of the application subtasks to the processors in the heterogeneous parallel hardware platform using previously stored off-line statically determined mappings. In particular, it was shown how the HC Kernel can be used to create the MDDG Table off-line and then use it to make real-time on-line decisions and selections of mappings. In addition to the HC Kernel, the relevant parts of other components of the IOS were briefly discussed. The overall strategy of the IOS and the interactions of the IOS components are summarized in Figures 1 to 3. The HC Kernel ideas introduced here can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the applications and platforms considered here.

The goal of this paper has been to introduce the concepts of the HC Kernel, explain when it can be applied, and to give a framework for developing it. Future research, by our team and other groups, includes building HC Kernel based systems for a variety of different applications and a variety of different target hardware platforms to determine where the HC Kernel methodology can be best exploited.

Acknowledgments

The authors thank Janet M. Siegel, Min Tan, Tracy Braun, Ricky Yu-Kwong Kwok, and the anonymous reviewers for their comments. A preliminary version of this paper was presented at the *6th Heterogeneous Computing Workshop.*

This research is supported by the Army Research Laboratory, under contract number DAAL01-96-C-0031. H. J. Siegel is a consultant for Architecture Technology Corporation.

References

1. S. Brandt and J. R. Budenske, "Starcon—a reconfigurable fieldable signal processing system," *SPIE Conf. on Image Understanding in the '90s: Building Systems that Work, SPIE Vol. 1406*, pp. 122–126, Oct. 1990.

2. J. R. Budenske, H. J. Siegel, R. S. Ramanujan, K. J. Thurber, and M. D. Pritt, *Intelligent Operating System Final Technical Report*, Architecture Technology Corp., Minneapolis, MN, Technical Report ATC-RD-96-04, 91 pp., Oct. 1996.
3. J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "'Modeling ATR applications for intelligent execution upon a heterogeneous computing platform," *1997 IEEE International Conf. on Systems, Man, and Cybernetics (SMC '97)*, pp. 649–656, Oct. 1997.
4. C. H. Chu, E. J. Delp, L. H. Jamieson, H. J. Siegel, F. J. Weil, and A. B. Whinston, "A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture," *Journal of Parallel and Distributed Computing*, Vol. 6, No. 3, pp. 598–622, June 1989.
5. P. David, S. Balakirsky, and D. Hillis, "A real-time automatic target acquisition system," *Conf. on Unmanned Vehicle Systems*, pp. 183–198, July 1990.
6. P. David, P. Emmerman, and S. Ho, "A scalable architecture system for automatic target recognition," *13th AIAA/IEEE Digital Avionics Systems Conf.*, pp. 414–420, Oct. 1994.
7. L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, 1991.
8. T. H. Einstein, "Mercury Computer Systems' modular heterogeneous RACE multicomputer," *6th Heterogeneous Computing Workshop (HCW '97)*, pp. 60–71, Apr. 1997.
9. M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.
10. D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 11, pp. 1427–1436, Nov. 1989.
11. R. F. Freund, "The challenges of heterogeneous computing," *Parallel Systems Fair at the 8th Int'l Parallel Processing Symp.*, pp. 84–91, Apr. 1994.
12. R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for heterogeneous computing," *2nd Int'l Symp. on Parallel Architectures, Algorithms, and Networks (I-SPAN '96)*, pp. 514–521, June 1996.
13. A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, pp. 78–86, June 1993.
14. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
15. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, MI, 1975.
16. M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95)*, pp. 93–100, Apr. 1995.
17. A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, pp. 18–27, June 1993.
18. M. B. Kulaczewski and H. J. Siegel, "Implementations of a feature-based visual tracking algorithm on two MIMD machines," *1997 International Conf. on Parallel Processing (ICPP '97)*, pp. 431–440, Aug. 1997.
19. J. L. Ribeiro Filho and P. C. Treleaven, "Genetic-algorithm programming environments," *IEEE Computer*, Vol. 27, No. 6, pp. 28–43, June 1994.
20. P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th Heterogeneous Computing Workshop (HCW '96)*, pp. 98–117, Apr. 1996.
21. H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping computer-vision-related tasks onto reconfigurable parallel processing systems," *IEEE Computer*, Vol. 25, No. 2, pp. 54–63, Feb. 1992.
22. H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. Alexander Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, ed. A. Y. Zomaya, McGraw-Hill, New York, NY, pp. 725–761, 1996.
23. H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, ed. A. B. Tucker, Jr., CRC Press, Boca Raton, FL, pp. 1886–1909, 1997.
24. H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW '96)*, pp. 86–97, Apr. 1996.

25. M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, pp. 17–26, June 1994.
26. V. S. Sunderam, "Design issues in heterogeneous network computing," *Workshop on Heterogeneous Processing*, revised edition, pp. 101–112, Mar. 1992.
27. Y. G. Tirat-Gefen and A. C. Parker, "MEGA: An approach to system-level design of application specific heterogeneous multiprocessors," *5th Heterogeneous Computing Workshop (HCW '96)*, pp. 105–117, Apr. 1996.
28. 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," *Journal of Parallel and Distributed Computing*, Special Issue on Parallel Evolutionary Computing, Vol. 47, No. 1, pp. 8–22, November 1998.