Mapping Tasks onto Heterogeneous Computing Systems

Howard Jay Siegel and Muthucumaru Maheswaran
{hj@purdue.edu, maheswar@ecn.purdue.edu}
Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285, USA

Abstract -- The goal of this invited tutorial paper is to provide an overview of three current research efforts in heterogeneous computing that focus on methods for determining a mapping of an application onto a heterogeneous suite of machines. The first study involves a genetic-algorithm approach for mapping the subtasks of an application task onto the machines in a distributed heterogeneous system. This is a static compile-time approach that must be used off-line (prior to task execution) due to its long run time. The second topic is the high-level design of components of an intelligent operating system for mapping and dynamically remapping automatic target recognition tasks onto a heterogeneous parallel platform. The intelligent operating system uses a new technique for dynamically selecting new mappings on-line during task execution from among choices precomputed off-line. Last, some initial preliminary results from a new research project for designing a dynamic mapping heuristic that does not use precomputed mappings is described. This dynamic heuristic is fast and is suitable for operation during application execution. Future research directions are discussed for all three projects.

Keywords: automatic target recognition, distributed computing, genetic algorithms, heterogeneous computing, mapping, matching, MSHN, parallel processing, scheduling.

1. Introduction

High-performance computers typically achieve only a fraction of their peak capabilities on certain portions of some application tasks. This is because different subtasks of an application can have very different computational requirements that result in needs for different machine capabilities. A single machine architecture cannot satisfy all the computational requirements in certain applications equally well. Thus, the use of a heterogeneous computing environment is more appropriate.

The research presented in Section 2 was supported by NRaD under subcontract number 20-950001-70; the research presented in Section 3 was supported by the Department of Defense, Small Business Innovative (SBIR) Program, funded by the Army Research Laboratory, under contract number DAAL01-96-C-0031; and the research presented in Section 4 is supported by the DARPA/ITO Quorum Program under NPS subcontract number N62271-97-M-0900.

One type of heterogeneous computing (HC) system provides a variety of architectural capabilities, orchestrated to perform an application whose subtasks have diverse execution requirements [SiA96]. A mixed–machine system is a heterogeneous suite of different types of independent machines interconnected by a high-speed network. When the goal is to perform an application task, each subtask must be matched to the machine that results in the lowest overall task execution time. To exploit HC systems in such situations, a task must be decomposed into subtasks, where each subtask is computationally homogeneous, and different subtasks may have different machine architectural requirements. These subtasks may share initial or generated data, creating the potential for inter-machine data transfer overhead.

Examples of applications that have been implemented on mixed-machine HC systems include a simulator for mixing in turbulent convection [KlM93], the interactive rendering of multiple earth science data sets on the CASA testbed [Spe90], and the computation of the radiation treatment planning for cancer patients on VISTAnet [RoC92]. Typically, users must perform task decomposition and subtask matching and scheduling. When performing matching and scheduling, the user must consider machine availability, the way in which each subtask's computational requirements match each machine's capabilities, inter-machine shared data transfers, and others factors [SiA96]. One long-term pursuit in HC is to do decomposition, matching, and scheduling automatically [SiD97].

This invited tutorial paper provides an overview of three current research efforts in distributed and parallel mixed-machine HC. In particular, this tutorial paper focuses on examples of methods for determining a mapping of an application onto a heterogeneous suite of machines (i.e., a matching of application subtasks to machines and a scheduling for the execution order and inter-machine communications of these subtasks). A brief summary of each project is given, including references for more complete information if available. Possible future research directions for each project are also given. A broader view of the HC field and related open research problems can be found in [Esh96, FrS93, KhP93, SiA96, SiD97].

The first study, presented in Section 2, involves a genetic-algorithm approach developed at Purdue University for mapping the subtasks of an application task onto the machines in a distributed heterogeneous system [WaS98]. This is a static (i.e., compile-time) approach that must be used off-line (prior to task execution) due to its long run time.

In some situations, the execution characteristics of an application task may not be derivable prior to run time and changes during run time as a function of the input data. When this occurs, a single static mapping may not be effective over a long time period, and dynamic mapping and remapping schemes should be considered, such as the two

approaches discussed next.

The second topic, explored jointly by Purdue University and Architecture Technology Corporation, is the high-level design of components of an intelligent operating system for mapping and dynamically remapping automatic target recognition tasks onto a heterogeneous parallel platform [BR97a, BR97b]. The intelligent operating system uses a new technique for dynamically selecting new mappings on line during task execution from among choices precomputed off line. Section 3 provides an overview of this work.

Section 4 describes some initial preliminary results from a new Purdue research project for designing a dynamic mapping heuristic that does not use precomputed mappings. This dynamic heuristic is fast and is suitable for operation during application execution. It is based on a list-scheduling type of algorithm.

The goal of this invited tutorial paper is to introduce the reader to these three current approaches to mapping research in the field of HC. The reader is encouraged to learn more about them, and other research topics in HC, from the references cited. Furthermore, the reader is encouraged to consider examining the future research problems listed in the following sections and in [KhP93, SiA96, SiD97, Sun92].

2. A Genetic-Algorithm Approach for Task Mapping

In general, the problem of performing matching and scheduling in an HC environment is NP-complete [Fer89], and therefore some heuristic must be employed. As an example of current HC research on mapping statically (i.e., at compile time), a genetic-algorithm approach from [WaS98] is summarized. An application task is decomposed into a set of subtasks S of size $|S|$. Let s_i be the i-th subtask. An HC suite consists of a set of machines M. Let m_j be the j-th machine. Each machine can be a different type. The global data items are data items that need to be transferred between subtasks.

The following assumptions about the applications and HC environment are made. Each application task will be represented by a DAG (directed acyclic graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the the data dependencies between subtasks. (Note that while the subtasks' dependencies are represented as a DAG, subtasks themselves may contain loops.) For each global data item, there is a single subtask that produces it (producer) and there are some subtasks that need this data item (consumers). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred over it. This task has exclusive use of the HC environment, and the genetic-algorithm-based mapper controls the HC machine suite (hardware platform). Subtask execution is non-preemptive. The estimated expected

execution time of each subtask on each machine is known. For each pair of machines in the HC suite, an equation for estimating the time to send data between those machines as a function of data set size is known.

Genetic algorithms (GAs) provide a promising heuristic approach to optimization problems that are intractable [Dav91, Gol89, Hol75, RiT94, SrP94]. The first step is to encode some of the possible solutions as chromosomes, the set of which is referred to as a population. In the [WaS98] approach, each chromosome consists of two parts: the matching string and the scheduling string. Let mat be the matching string, which is a vector of length $|S|$. If $\text{mat}(i) = j$, then subtask s_i is assigned to machine m_j. Typically, multiple subtasks will be assigned to the same machine, and then executed in a non-preemptive manner based on an ordering that obeys the precedence constraints (data dependencies) specified in the task DAG. The scheduling string is a topological sort of the DAG representing the task (i.e., a valid total ordering of the partially ordered DAG). Define ss to be the scheduling string, which is a vector of length $|S|$. If $\text{ss}(k) = i$, then subtask s_i is the k-th subtask in the total ordering. Because it is a topological sort, if subtask $\text{ss}(k)$ is a consumer of a global data item produced by subtask $\text{ss}(j)$, then $j < k$. The scheduling string gives an ordering of subtasks that is used by the evaluation step.

Each chromosome is associated with a fitness value, which is the completion time of the solution (i.e., mapping) represented by this chromosome (i.e., the expected execution time of the application task if the mapping specified by this chromosome were used). Overlapping among all of the computations and communications performed is limited only by inter-subtask data dependencies and the availability of the machines and the inter-machine network.

In the initial population generation step, a predefined number of distinct chromosomes are randomly created. The solution from a non-evolutionary heuristic is also included in the initial population. After the initial population is determined, the genetic algorithm iterates until a predefined stopping criterion is met. Each iteration consists of the selection, crossover, mutation, and evaluation steps.

In the selection step, some members of the population are removed and others are duplicated. First, all of the chromosomes in the population are ordered (ranked) by their fitness values. Then a rank-based roulette wheel selection scheme is used to implement proportionate selection [Hol75]. The population size is kept constant and a chromosome representing a better solution has a higher probability of having one or more copies in the next generation population. This GA approach also incorporates elitism, i.e., the best solution found so far is always maintained in the population [Rud94].

The selection step is followed by the crossover step, where some chromosomes are

paired and corresponding components of the paired chromosomes are exchanged. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings in the current population. For each pair, it randomly generates a cutoff point, and divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in the bottom part of one string is the relative positions of these subtasks in the other original scheduling string, thus guaranteeing that the newly generated scheduling strings are valid schedules. The crossover operator for the matching strings randomly chooses some pairs of the matching strings in the current population. For each pair, it randomly generates a cutoff point, and divides both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The next step is mutation. The scheduling string mutation operator randomly chooses some scheduling strings in the current population. Then for each chosen scheduling string, it randomly selects a victim subtask. The valid range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed and still have a valid topological sort of the DAG. The victim subtask is moved randomly to another position in the scheduling string within its valid range. The matching string mutation operator randomly chooses some matching strings in the current population. For each chosen matching string, it randomly selects a subtask entry. Then the machine assignment for the selected entry is changed randomly to another machine.

The last step of an evolution iteration is the evaluation step to determine the fitness value of each chromosome in the current population. A communication subsystem that is modeled after a HiPPI LAN with a central crossbar switch [ToR93] was assumed for the tests that were conducted. As stated earlier, the above steps of selection, crossover, mutation, and evaluation are repeated until one of the stopping criteria are met: (1) the number of iterations reaches some limit (e.g., 1000), (2) the population converged (all the chromosomes had the same fitness value), or (3) the best solution found was not improved after some number of iterations (e.g., 150).

In the tests of this GA approach in [WaS98], simulated program behaviors were used. Small-scale tests were conducted with up to ten subtasks, three machines, and population size 50. For each test, the GA approach found a solution (mapping) that had the same expected completion time for the task as that of the optimal solution found by exhaustive search. Larger tests with up to 100 subtasks, 20 machines, and population size 200 were conducted. This GA approach produced solutions (mappings) that averaged from 150% to 200% better than those produced by the non-evolutionary levelized min-time (LMT) heuristic proposed in [IvO95]. The heuristic in [IvO95] was selected for comparison

because it used a similar model of HC. The GA approach took much more time to generate the mappings than did the LMT approach; however, if the mappings are being done at compile time for production tasks that will be executed repeatedly, the generation time is worthwhile.

There are number of ways this GA approach for HC task mapping may be built upon for future research. These include extending this approach to allow multiple producers for each of the global data items, parallelizing the GA approach, developing evaluation procedures for other communication subsystems, and considering loop and data-conditional constructs that involve multiple subtasks. This last extension will be particularly difficult to handle at compile time if the loop bounds and data-conditional constructs are input-data dependent.

3. Dynamic Use of Statically-Derived Mappings

This section, which provides a simplified summary of [BR97a], focuses on a particular application domain (iterative automatic target recognition (ATR) tasks) and an associated specific class of dedicated HC hardware platforms. The contribution of [BR97a] is that, for the computational environment considered, it presents a methodology for on-line (i.e., execution-time) input-data dependent remapping of the application subtasks to the processors in the HC hardware platform using one of a set of previously stored off-line (i.e., statically) determined mappings. That is, the operating system will be able to decide during the execution of the application whether or not 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 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 for enabling the on-line use of off-line mappings is called the IOS (Intelligent Operating System). The ATR Kernel component makes decisions on how a given ATR application task should be performed, including determining the partial ordering of subtasks and which algorithms should be used to accomplish each subtask [BR97b]. The HC Kernel component decides 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 component (i.e., the low level operating system) to execute the application and monitor its execution. Thus, the ATR Kernel deals with application issues, while the HC Kernel deals with implementation issues. Information from the Algorithm Database and the Knowledge

Base is used to support the ATR and HC Kernels.

The IOS has not been implemented; such an implementation is a major undertaking and outside the scope of [BR97a], which is the design concepts for the HC Kernel. The IOS design has its roots in the high-level model presented in [ChD89] for automatic dynamic processor allocation in a partitionable parallel machine with homogeneous processors. The IOS differs from other real-time HC mapping techniques in that it allows on-line execution-time use of off-line precomputed mappings. This is significant because off-line heuristics can produce better mappings as a result of much longer execution times to search for a good solution than what is practical for an on-line heuristic. Thus, the mapping quality of a time-consuming off-line heuristic can be approached at real-time speeds.

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, i.e., applications that will iteratively execute a DAG such as described in Section 2. The expected number of subtasks is ten to 50. It is assumed that each ATR application in the class under consideration 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 [DaB90] is an example of such an iterative ATR application.

The type of target hardware platforms considered for this study are driven by the expected needs of the kinds of ATR applications that are of interest to the U.S. Army Research Laboratory (e.g., [DaE94]). For the intended application environment, it is assumed that there will be up to four different types of processors (e.g., SHARC, PowerPC), and up to a total of 64 processors (of all types combined). These processors will comprise the heterogeneous parallel architecture onto which application tasks will be mapped. The IOS approach is appropriate for larger HC platforms as well.

For the initial iteration through the set of subtasks, the IOS will employ user-provided information about the processing environment in its selection of algorithms for the subtasks, and their associated implementations. As part of this, the IOS will choose a precomputed mapping to decide how to assign one or more processors to each subtask.

Dynamic parameters are characteristics of the given application that may change during run time and can be computed by the application as it executes, as a function of the input data. The values of dynamic parameters may in^{a}uence the decision of how to map a task onto the HC platform, and even initiate the use of a different algorithm for a given subtask. Examples of dynamic parameters include: (1) amount of clutter found in the current image of a sequence and (2) number of objects currently located that need to be

identified. 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.

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, 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 the 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.

To provide the HC kernel with the decision capability and off-line mappings it will need, the IOS has a Scenario Generator, which is the component of the IOS that uses information provided by the application developer to derive representative values for these dynamic parameters. 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, $0 \le i < D$. Each set of D values for these D dynamic parameters, one per parameter, is called a scenario. The number of different scenarios that can be generated is $\sigma = C_0 \times C_1 \times \cdots \times C_{D-1}$. It is important to pick an effective set of dynamic parameters and associated representative choices for parameter values, and to keep $|\sigma|$ at a reasonable size. For each scenario, an off-line heuristic, such as a version of the GA described in Section 2, is used to generate a mapping that is stored in a D-dimensional array in the Knowledge Base and indexed by the dynamic parameter values for that scenario. The expected execution time of the task for that given scenario and mapping is also stored.

As the application is executing, the HC Kernel monitors the run-time values of the dynamic parameters at the end of each iteration through the underlying DAG to decide whether to continue with the current mapping, or to select and implement a new mapping (for the next iteration) from among the off-line generated mappings. It does this by finding the scenario whose associated dynamic parameter values are closest to the actual dynamic parameter values at the end of the iteration. It then compares the previously stored execution time for the new mapping associated with that scenario to perform an iteration of the DAG versus the actual time for the last iteration plus an estimated reconfiguration time. Thus, the off-line processing provides a set of predetermined mappings that the on-

line processing can index in real time.

The IOS 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 include sensor-based robotics, intelligent vehicle highway systems, air traffic control, nuclear facility maintenance, weather prediction, intruder detection, and manufacturing inspection.

Future work based on this IOS involves actual implementation of the components discussed and a graphical user interface. The Algorithm Database and Knowledge Base must be populated with the information needed for the IOS to be a functioning ATR system for some set of applications. Implementation of the IOS will require the study of various research problems, such as: how to select the number of scenarios to use in a given environment; how to facilitate the selection of dynamic parameters and their representative values; how to effectively merge multiple ATR tasks to execute simultaneously on the same HC platform and be dynamically remapped as required by the IOS (even when the iteration times for the tasks are different); and how to reasonably estimate the remapping reconfiguration overhead time as a function of the application and platform. Another area for investigation is how to use this technique in broader environments where the dynamic parameters correspond to general computational properties rather than being a function of the application.

4. Dynamically-Derived Mappings

Preliminary results from a new research project on a dynamic remapping heuristic for HC systems is summarized in this section. The heuristic described here uses current system information (e.g., machine loading at run time) to improve a an initial static mapping by periodic dynamic remapping. As in Sections 2 and 3, it is assumed that the subtasks of the application(s) are represented by a DAG and the dynamic-heuristic-based mapper controls the HC machine suite. The dynamic mapper executes on a dedicated workstation that is tightly coupled with the HC machine suite. The scheduler allows multiple applications to execute concurrently on the HC machine suite, i.e., at any given time the subtasks executing on the system can be from different applications.

The dynamic heuristic executes in two phases. In the first phase, a priority is computed for each subtask from the static mapping that is provided to the algorithm (i.e., any static mapping can be used). During this phase, which is executed at compile time, the DAG is level sorted into m level sets, numbered consecutively from 0 to $m-1$. The level sorting clusters the subtasks into level sets such that the subtasks within a level set are

independent (i.e., there are no direct data dependencies among the subtasks within a level). Furthermore, for the j-th subtask at level k, $s_{j,k}$, there exists at least one incident edge (data dependency) such that the source subtask is at level $k-1$, i.e., an incident edge from some $s_{i,k-1}$. The level set at level $m-1$ includes only those subtasks without any descendents and at level 0 includes only those subtasks without any predecessors.

The second phase executes in real time with the execution of the subtasks. In this phase, dynamic remappings are performed if such remappings are expected to yield a performance benefit. The dynamic remappings are non-preemptive and they involve updating the mapping for the next level before the execution of that level begins.

The priority assignment begins with the level set at level $m-1$ and assigns a priority to each subtask within the level set. The priority of each subtask in the $(m-1)$-th level set is its expected computation time on the machine it was assigned by the static matching. Now consider the k-th level, $0 \le k < m-1$. Let e_i be the expected computation time of the subtask $s_{i,k}$ for the given static matching, c_{ij} be the communication time for a descendent $s_{j,q}$ of $s_{i,k}$ to get all the relevant data items from $s_{i,k}$ (where $q \ge k+1$), priority($s_{i,k}$) be the priority of the subtask $s_{i,k}$, and set of descendents of $s_{i,k}$ be the set of subtasks in level(s) $q \ge k+1$ such that each subtask is dependent on a data item generated by subtask $s_{i,k}$. With the above definitions, the priority of a subtask $s_{i,k}$ is given by:

$$\text{priority}(s_{i,k}) = e_i + \max_{s_{j,q} \in \textit{set of decendents}} (c_{ij} + \text{priority}(s_{j,q})).$$

The priorities of the subtasks in other level sets can be computed using a similar recursion. The priority of a subtask can be interpreted as the length of the critical path from the point the given subtask is located on the DAG to the end of the execution of all its descendents. The dynamic remapping heuristic is based on the idea that by executing the subtasks with higher priorities as fast as possible it is possible to expect a shorter completion time for the application.

The execution of the subtasks of an application proceeds from level 0 to level $m-1$. Consider the situation where the dynamic heuristic is trying to remap the subtasks at the i-th level while the subtasks at the $(i-1)$-th level are being executed. The dynamic heuristic starts remapping the i-th level subtasks when the first $(i-1)$-th level subtask begins its execution. When level i is being scheduled, it is highly likely that actual execution time information can be used for most subtasks from levels 0 to $i-2$. There may be some long-running subtasks from levels 0 to $i-2$ that could be still executing when subtasks from level i are being considered for remapping. For such subtasks expected execution times are used. The dynamic heuristic examines the subtasks in the i-th level set in descending order based on their precomputed static priorities. The subtask is assigned to the machine that gives the shortest partial completion time for the subtasks that

have been scheduled (including this subtask).

A simulator was implemented to evaluate the performance of the remapping heuristic versus a static mapper called the baseline heuristic [WaS98]. The baseline heuristic initially uses level sorting to levelize the subtasks. Then all subtasks are ordered such that a subtask at level i comes before one at level j, $i < j$. The subtasks in the same level are arranged in descending order based on the number of descendents of each subtask (ties broken arbitrarily). The subtasks are considered for assignment in this order. The subtask is assigned to the machine that gives the shortest partial completion time for the subtasks that have been scheduled (including this subtask).

The inputs to the simulator are the randomly generated DAG, the static mapping, the percentage deviation in the parameters (subtask execution times or communication times), and the machine and network information. The percentage deviation is defined as the percentage by which the simulated expected time differs from the simulated actual time. If α is the percentage deviation and u a random number that is uniformly distributed in [0,1], then the simulated actual value of a parameter x can be modeled as $x(100-\alpha+2\alpha u)/100$. The simulator evaluates the execution of the static mapping by computing the completion times of the subtasks using the simulated actual parameter values and the static mapping. The remapped execution is simulated by computing the completion times for the simulated actual parameter values and the dynamically remapped mapping.

The simulation results are shown in Figure 1 for ten machines and 100 subtasks. Each data point is an average of ten simulation runs. The average simulation time for each run of this experiment was 0.62 seconds and the average time for executing the remapping heuristic per level of the task graph was 0.0041 seconds, with a minimum of 0.00312 seconds and a maximum of 0.00764 seconds. As the percentage deviation of the parameters from their static estimates increases, the difference between the task completion time using the "static mapping only" versus the task completion time using the "dynamic remapping" increases. Thus, dynamic remapping can improve the performance of an initial static mapping. However, the time taken by the dynamic heuristic to remap the i-th level of the task graph must be less than the time difference between the time a $(i-1)$-th level subtask started execution and the time the machine and data necessary to start the execution of an i-th level subtask becomes available; otherwise, the dynamic heuristic may delay the execution of an i-th level subtask.

Other groups have also studied dynamic mapping heuristics for HC systems (e.g., [FrC97, HaL95, LeP95]). Methods presented in these papers are different from the work presented here. Future work involves implementing their algorithms in the simulator

discussed here to compare the performance of the dynamic heuristic presented here with those presented in the above papers.

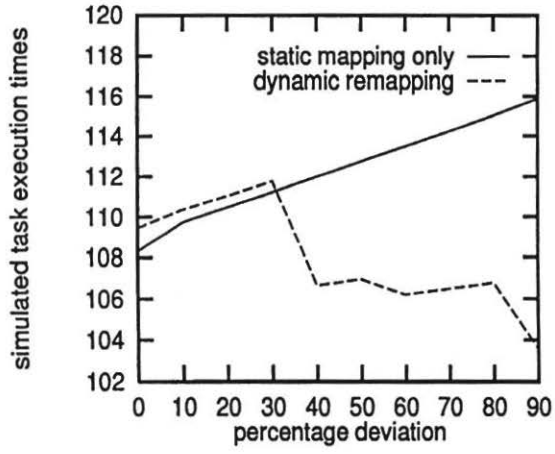

Figure 1: Simulation results for the dynamic heuristic for ten machines and 100 subtasks.

The following are some areas for future research for the dynamic heuristic. More simulation data should be collected, e.g., 100 test runs per data point compared to the ten runs used for the initial study. Simulation studies should be conducted to compare the performance of a dynamic remapper with a static GA-based mapper (such as described in Section 2) under varying parameter values. In particular, because the GA-based mapper has been shown to provide mappings superior to the much faster baseline heuristic when the expected execution times are used as actual times [WaS98], it will be interesting to examine at what percentage variation the dynamic remapping makes a significant improvement over the GA (this can be done using the baseline or the GA as the initial static mapper). Other areas include investigating the use of alternate level set definitions and priority computation schemes, and exploring ways of improving the dynamic heuristic to support multiple data-copies (a situation where a subtask can have multiple sources (machines) for a needed data item) [TaS97].

5. Conclusions

A novel genetic-algorithm approach for task matching and scheduling in HC environments was presented in Section 2 [WaS98]. It is applicable to the static scheduling of production jobs and can be readily used for scheduling multiple independent tasks (and their subtasks) collectively. For small-scale tests, the proposed approach found optimal

solutions. For larger tests, it outperformed a published non-evolutionary heuristic.

In Section 3, for the computational environment considered, an HC Kernel was described for making real-time on-line input-data-dependent remappings of the application subtasks to the processors in the HC hardware platform using previously stored off-line statically determined mappings (e.g., using a GA) [BR97a]. The IOS ideas can also be used for other application domains and other HC hardware platforms whose characteristics are similar to those considered there.

A different approach to dynamic remapping was discussed in Section 4. Here the actual mapping for one set of subtasks is computed concurrently with the execution of an earlier set of subtasks. When a subtask is mapped, any available run-time information about the system state is used by the mapping heuristic. This research is just beginning, and the initial results are very promising.

Some of the future research outlined at the ends of Sections 2 to 4 may be pursued as part of a DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort that includes NPS (Naval Postgraduate School), NRaD (a Naval Laboratory), Purdue, and USC (University of Southern California). It builds on SmartNet, an operational scheduling framework and system for managing resources in a heterogeneous environment developed at NRaD [FrK96]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

HC is an exciting research field with practical uses. The reader in encouraged to investigate the research problems listed at the ends of Sections 2 to 4 and to explore this field further using the references cited.

Acknowledgments -- The authors thank Janet M. Siegel for her comments on this paper. A version of some of this material also appeared in an invited keynote paper for the 1997 Parallel and Distributed Processing Techniques and Applications Conference.

References

[BR97a] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "On-line use of off-line derived mappings for iterative automatic target recognition tasks and a particular class of hardware platforms," *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 96-110.

[BR97b] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "Modeling ATR applications for intelligent execution upon a heterogeneous computing platform," *1997 IEEE Int'l Conf. Systems, Man, and Cybernetics (SMC '97),*

Oct. 1997, to appear.

[ChD89] 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," *J. of Parallel and Distributed Computing,* Vol. 6, No. 3, June 1989, pp. 598-622.

[DaB90] P. David, S. Balakirsky, and D. Hillis, "A real-time automatic target acquisition system," *Conf. on Unmanned Vehicle Systems,* July 1990, pp. 183-198.

[DaE94] P. David, P. Emmerman, and S. Ho, "A scalable architecture system for automatic target recognition," *13th AIAA/IEEE Digital Avionics Systems Conf.,* Oct. 1994, pp. 414-420.

[Dav91] L. Davis, ed., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, NY, 1991.

[Esh96] M. M. Eshaghian, ed., *Heterogeneous Computing,* Artech House, Norwood, MA, 1996.

[Fer89] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. on Software Engineering,* Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.

[FrC97] R. F. Freund, B. R. Carter, D. Watson, E. Keith, F. Mirabile, and H. J. Siegel, "Generational scheduling for heterogeneous computing systems," *Information Science Journal,* Special Issue on Parallel and Distributed Processing Techniques and Applications, accepted and scheduled to appear in 1997.

[FrK96] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for meta-computing," *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96),* June 1996, pp. 514-521.

[FrS93] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer,* Special Issue on Heterogeneous Processing, Vol. 26, No. 6, June 1993, pp. 13-17.

[Gol89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, Reading, MA, 1989.

[HaL95] B. Hamidzadeh, D. J. Lilja, and Y. Atif, "Dynamic scheduling techniques for heterogeneous computing systems," *Concurrency: Practice and Experience,* Vol. 7, No. 7, Oct. 1995, pp. 633-652.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems,* Univ. of Michigan Press, Ann Arbor, MI, 1975.

[IvO95] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 93-100.

[KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer,* Vol. 26, No. 6, June 1993, pp. 18-27.

[KlM93] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, "A case study in metacomputing: Distributed simulations of mixing in turbulent convection," *2nd Workshop on Heterogeneous Processing (WHP '93),* Apr. 1993, pp. 101-106.

[LeP95] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30-34.

[RiT94] J. L. Ribeiro Filho and P. C. Treleaven, "Genetic-algorithm programming environments," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 28-43.

[RoC92] J. Rosenman and T. Cullip, "High-performance computing in radiation cancer treatment," *CRC Critical Reviews in Biomedical Engineering*, Vol. 20, 1992, pp. 391-402.

[Rud94] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, Vol. 5, No. 1, Jan. 1994, pp. 96-101.

[SiA96] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[Spe90] "Special report: Gigabit network testbeds," *IEEE Computer*, Vol. 23, No. 9, Sep. 1990, pp. 77-80.

[SrP94] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17-26.

[Sun92] V. S. Sunderam, "Design issues in heterogeneous network computing," *Workshop on Heterogeneous Processing (WHP '92)*, revised edition, Mar. 1992, pp. 101-112.

[TaS97] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE Trans. on Parallel and Distributed Systems*, accepted and scheduled to appear in 1997.

[ToR93] D. Tolmie and J. Renwick, "HiPPI: Simplicity yields success," *IEEE Network*, Vol. 7, No. 1, Jan. 1993, pp. 28-32.

[WaS98] 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. of Parallel and Distributed Computing*, Special Issue on Parallel Evolutionary Computing, accepted and scheduled to appear in 1998. (A preliminary version appeared in the *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 72-85.)