A Framework for Automated Software Partitioning and Mapping for Distributed Multiprocessors

Ranga S. Ramanujan, Jordan C. Bonney, and Kenneth J. Thurber
Architecture Technology Corporation, Minneapolis, MN, USA

Rakesh Jha
Honeywell Technology Center, Minneapolis, MN, USA

Howard Jay Siegel
Purdue University, E.C.E. School, West Lafayette, IN, USA

Abstract

One of the major impediments to the widespread use of large-scale, distributed memory multiprocessors is the difficulty of efficiently partitioning and mapping application algorithms onto these machines so as to extract a large portion of the machines’ peak performance. In this paper, we present the preliminary accomplishments of an ongoing effort aimed at automating the complex tasks of software partitioning and mapping during the system definition phase of application development for distributed memory multiprocessors. We describe a technique called the Augmented Task Dependency Graph (ATDG) for representing the high-level design of the application software. The ATDG allows one to express functional parallelism as well as data parallelism in a manner that facilitates automated partitioning and mapping. We propose a new strategy for searching through the possible space of design choices for partitioning and mapping. The proposed approach, called hierarchical hybrid search, organizes the search space as a hierarchy of sub-spaces. It permits the use of different search techniques for searching through different search sub-spaces. Examples of search techniques that could be employed in the proposed approach include hill-climbing, simulated annealing, and genetic algorithms.

1. Introduction

Today’s parallel computers cover a wide spectrum of architectural approaches. At one end of the spectrum, there are machines composed of thousands of relatively simple processors. At the other end of the spectrum, are "vector multiprocessors" which consist of a small number of much more complex processors, each built using the fastest available technology and the most sophisticated architecture. In between the two extremes are a wide variety of parallel machines that employ a large number of inexpensive computing nodes connected together by a scalable interconnection network. Each computing node in such an architecture may consist of an inexpensive microprocessor and a memory element. These machines are generally referred to as distributed memory multiprocessors (or distributed multiprocessors, for short) or MIMD machines [1]. It is this category of high-performance computers that is of interest to this research effort. Examples of commercially available machines of this kind are Intel Paragon, nCUBE, Cray T3D, and IBM SP2.

The potential of distributed multiprocessors to deliver cost-effective solutions to problems that require high-performance computers is well recognized. Despite the tremendous price/performance advantage offered by distributed multiprocessors, they are yet to make major inroads into mainstream high-performance computing. One of the major impediments to the widespread use of distributed multiprocessors is the difficulty of efficiently partitioning and mapping algorithms onto these machines so as to extract a large portion their peak performance [2].

Software partitioning and mapping are critical tasks during system development because the performance of an

---

1This material is based upon work supported by the National Science Foundation’s Small Business Innovative Research (SBIR) program under award number DMI-9461684. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.
application implemented on a parallel computer depends on how well its software is partitioned and mapped on the parallel machine. A poor partitioning of the application software or a poor allocation of the application fragments to the hardware elements may lead to unacceptable performance regardless of the compute power of the underlying hardware.

In some embedded systems, processors are hardwired to sensors and actuators. In such a structured environment, the criteria for partitioning and mapping of applications to the multiprocessor are fairly obvious. However, in many applications where multiprocessors are used, the component processors provide an undifferentiated computational resource, i.e., they are functionally homogeneous. In this case, currently, designers of applications for multiprocessor systems must specify the partitions of the application software and manually assign the partitions to the component processors. The goal of the system designer is to select a partitioning and mapping solution that best meets the selection criteria for the application in question. Examples of such selection criteria for a partitioning and mapping solution include "a solution that minimizes execution time" and "a solution that minimizes the size of the multiprocessor system while ensuring that the execution time for the application does not exceed a given threshold."

For non-trivial problems, the number of possible partitions and mappings that must be searched to arrive at an acceptable solution is far too large to be performed manually. It is therefore essential to automate this task. Even if this task were automated, exhaustive search of all possible partitions and mappings becomes computationally intractable. Some intelligence must therefore be built into the automation process to guide the search through this potentially enormous solution space for the partitioning and mapping problem.

Current research on automated software partitioning and mapping that addresses this problem generally assumes that the application code has already been written. The software description in the form of code is then analyzed by a tool (or tools) to generate a partitioning and mapping strategy for the code. Thus, current research on partitioning and mapping primarily focuses on the "back-end" of the parallel system development process, i.e., the implementation phase.

This research focuses on software partitioning and mapping as it applies to the "front-end" of the parallel system development process, i.e., the system definition or preliminary design phase. During the design phase, the system designer may have to evaluate various choices of algorithms and architectures. Code is written only after a promising algorithm/architecture combination is selected.

The system designer needs to perform an early evaluation of the design choices for a number of reasons. One reason may be to narrow the number of design choices to a manageable number. Attention can then be focused on these viable choices during the next stage of the development cycle, i.e., the detailed design phase where the choice may be furthered narrowed before actual code is developed for the selected algorithm and architecture combination. Another reason for the early evaluation of the design is to determine the size and cost of the parallel processing system needed for the application. This is especially important for embedded parallel systems.

There are three major technical challenges that must be addressed in automating the partitioning and mapping process:

1. The definition of an application modeling notation that can be used to capture the characteristics of the application software design at a high-level and that can be interpreted and manipulated by the partitioning and mapping tools.

2. The development of an architecture modeling notation that can be used to capture the salient characteristics of the target distributed memory parallel machine configurations for use by the automated partitioning and mapping process.

3. The design of intelligent strategies that can be used by the automated partitioning and mapping processes to search through the potentially enormous search space of machine configurations and software partitions and mappings to find a solution satisfying some user specified criteria.

Our on-going research project seeks to address each of the three challenges listed above. To address the first problem, we propose an application modeling notation called the augmented task dependency graph (ATDG) that is described in Section 2. To address the second problem, we propose a new architecture modeling notation that is presented in Section 3. The third problem above is addressed by a new strategy for searching through the possible space of design choices, called hierarchical hybrid search, that is described in Section 4. We conclude this paper with a discussion of related research and the status of our research effort.

2. Augmented Task Dependency Graph

The augmented task dependency graph (ATDG) notation that is described in this section captures the
characteristics of a high-level design of the application software. High-level software design implies that the software code implementing the computational tasks of the application may not be available at this time. The ATDG represents the structure of an application as an acyclic directed graph as shown in Figure 1. The nodes of the digraph represent computational tasks within the application, each of which may correspond to a code module in the final implementation of the application software. A pair of distinct tasks in the ATDG are connected by a directed arc if and only if the task at the head of the arc requires the results of the execution of the task at the tail of the arc. For example, in the ATDG depicted in Figure 1, the computational task labeled $T_2$ is dependent upon the results of task $T_1$.

The ATDG is thus a directed acyclic graph that specifies a partial ordering of the execution of the computational tasks of the application. This partial ordering of tasks also serves to specify the functional parallelism within the application. By functional parallelism we mean data-parallel task. The subtasks of a data-parallel task all perform the same computation but operate on different partitions of the data set associated with the original task. The parallel subtasks execute asynchronously, i.e., without instruction by instruction synchronization across the subtasks. This type of data parallelism is sometimes referred to as SPMD (Single Program Multiple Data-Streams) type parallelism [3]. The ATDG notation allows an application designer to identify data-parallel tasks in an application without specifying the actual partitioning of the task into sub-tasks. The selection of a specific partitioning of a data-parallel task into sub-tasks is performed by the automated partitioning and mapping process. Attributes for a data-parallel task in an ATDG are specified by the system designer to convey information that can be used by an automated partitioning tool to select a partitioning of the task.

The attributes associated with the tasks in an ATDG are listed and defined below. This is an initial list that can be extended and refined within the proposed framework.

**Task Identifier:** This uniquely identifies the task within the ATDG.

**Data Parallelism Indicator:** This is a boolean variable that indicates whether the task can be partitioned into parallel sub-tasks for exploiting inherent SPMD parallelism.

**Integer Operations:** This specifies the number of integer operations needed to perform the computations associated with the task.

**Floating Point Operations:** This specifies the number of floating point operations needed to perform the task.

**Data Set Size:** This indicates the size of the data associated with the computation.

**Data Transfer Volume:** This is the amount of data transferred by this task to all its successors in the ATDG (say in bytes).

The following additional task attributes only apply to data-parallel tasks. For other tasks, these attributes are ignored.

**Scaling Factors for Integer Operations (SFIOs):** These are real-valued constants $a$ and $b$ that are used by an automated partitioning tool to estimate the number of integer operations performed by the subtasks obtained for a specific partitioning of a data-parallel task. The number of integer operations performed by a subtask of a data-parallel task may sometimes be a function of the size of the data partition.

![Figure 1: Augmented Task Dependency Graph](image)

An ATDG may contain data-parallel tasks that can be partitioned into parallel subtasks. In Figure 1, task $T_3$ is a

![Diagrame](image)
assigned to the subtask. The number of integer operations performed by the subtask is given by \( a + b \cdot d \cdot t \) where \( d \) is the fraction of the partitioned data assigned to this sub-task for this partitioning (e.g., 0.2) and \( t \) is the total number of integer operations specified by the Integer Operations attribute of the data-parallel task.

Scaling Factor for Floating Point Operations (SFFO): This is an attribute similar to the previous attribute. This is used to estimate the number of floating point operations in a subtask for a specific partitioning of data.

Scaling Factor for Data Transfer (SFDT): This is a real number that is used by the automated partitioning and mapping process to estimate the amount of data transfer between the subtasks of a data-parallel task for a specific partitioning of the task. The inter-subtask data transfer volume is calculated as \( SFDT \cdot d \) where \( d \) is the fraction of the partitioned data that is assigned to this subtask.

The SFIO, SFFO and SFDT attributes of a data-parallel task are influenced by the data decomposition strategy used to partition the task (e.g., block decomposition, cyclic decomposition). Therefore, before specifying the attributes for a data-parallel task in an ATDG, the system designer must select a set of alternate data decomposition strategies that may be used for partitioning the task. For each candidate decomposition strategy, the system designer must specify the SFIO, SFFO, SFDT attributes associated with the data-parallel task.

As described above, the ATDG notation for modeling the high-level design of an application provides a way of identifying the inherent data parallelism within a computational task in addition to expressing functional parallelism among the tasks of an application.

3. Architecture Modeling Notation

The following paragraphs present an architecture modeling notation that can be used to capture the salient characteristics of the target distributed memory parallel machine configurations for use by the automated partitioning and mapping process. Before describing the model it is necessary to define certain terms. We use the term architecture or architecture type to refer to certain broad characteristics that may be shared by a number of parallel machines. A parallel machine configuration refers to a specific instance of a particular architecture type. Using an analogy from the uniprocessor word, the term "Unix workstation" may be viewed as defining an architecture type. Machine configurations corresponding to this type may be a Intel Pentium machine with 8 MB of memory running the Linux operating system. Note that there is not necessarily a one-to-one correspondence between a machine configuration and a computer system. The same computer system may be configured as another architecture type yielding a different machine configuration. For instance, the workstation described above may be configured to run Microsoft Windows. This yields a different machine configuration and architecture type (i.e., a Windows workstation) for the same computer system.

In the parallel computing world, architecture types may be defined on the basis of the parallel programming models, e.g., message passing using PVM or MPI versus globally shared address space. Examples of distinct machine configurations for the former type are a CRAY T3D configured to support PVM, and an IBM SP2 configured to support PVM.

In the architecture model defined by this effort, each machine configuration is characterized by a set of attributes that are listed and defined below. As with application attributes, the list below is an initial one that can be extended and refined within the proposed framework.

Architecture Type: This defines the type of architecture to which this machine configuration belongs.

Machine Size: This defines the maximum number of processors for this machine configuration.

Integer Operations Speed: This gives the speed with which each processor in the machine configuration processes integer operations.

Floating Point Operations Speed: This gives the speed with which each processor in the machine configuration processes floating point operations.

Data Transfer Speed: This defines the rate at which data units (e.g., bytes) are transferred by the machine between tasks/sub-tasks mapped on different processors.

Message Start-Up Latency: This defines the end-to-end latency encountered by messages exchanged between tasks residing on different processors of the machine configuration.

4. Hierarchical Hybrid Search Strategy

This section presents the design of an intelligent strategy that can be used by the automated partitioning and mapping processes to search through the potentially enormous search space of machine configurations and
software partitions and mappings to find a solution satisfying some user specified criteria. The need for an intelligent search strategy becomes apparent when one considers a typical usage scenario for the front-end design tools targeted by this effort. Consider the case of a system designer who has to define a parallel machine configuration for an embedded application. In choosing a machine configuration, the system designer must optimize two metrics: the execution time for the application and the size, weight, power consumption, and cost of the machine configuration.

Let us suppose that for the architecture type chosen for the application there are m possible choices for machine configurations, C₁, C₂, …., Cₘ. Let nᵢ represent the number of processors in machine configuration Cᵢ (for 1 ≤ i ≤ m). Let N be the total number of tasks in the ATDG representation of the application. Let D be the number of these tasks that are data-parallel. The maximum number of subtasks that any data-parallel task can be split into is nᵢ. Assume that there are P different partitioning strategies considered for each data-parallel task in the ATDG (e.g., block decomposition versus cyclic decomposition). Then for each Cᵢ, the total number of tasks and subtasks is

\[ tᵢ = (N-D) + nᵢD \]

The total number of different partitionings for each Cᵢ is

\[ pᵢ = P^D \]

Let us now consider the space of possible mappings of a partition pᵢ on machine configuration Cᵢ. Each processor in machine configuration Cᵢ may be assigned zero or more tasks/subtasks. The optimal mapping may use only a proper subset of the processors and assign multiple tasks/subtasks to those processors. This is because the performance metric we are using is the overall execution time of the application (not utilization), and, due to communication and synchronization overhead, the best execution time may be achieved by not using all available processors [3,4]. With tᵢ tasks/subtasks in the application partitioning for machine configuration Cᵢ and nᵢ processors in this machine configuration, the total number of possible mapping assignments is approximately

\[ \Sigma (nᵢ)^{tᵢ}pᵢ, \ 1 \leq i \leq m \]

To find the guaranteed optimal solution, the execution time resulting from each possible mapping for each partitioning of the application for each machine configuration will have to be evaluated. Obviously, for realistic size applications and machines, the complete search space of solutions cannot be evaluated exhaustively within a reasonable amount of time. Thus, an intelligent search strategy that quickly prunes the search space to find a near-optimal solution is required.

The intelligent search strategy developed by this effort is based upon the observation that the search space for the automated partitioning and mapping process can be decomposed into a hierarchy of search sub-spaces as shown in Figure 2. Each architecture type at the top of the hierarchy is associated with a set of machine configurations of that type. This set of machine configurations forms the search sub-space labeled S₁ in Figure 2. Each machine configuration in S₂ yields several partitions of the ATDG (or application software). The set of partitions of the application is represented by the search sub-space S₂ at the next lower level in the search hierarchy. Each partitioning of the application in S₂ yields a number of possible mappings for a machine configuration. The set of possible mappings for an application on a machine configuration forms the lowest level search sub-space, S₄, in the search hierarchy.

**Figure 2: Hierarchical Search Space for the Partitioning and Mapping Problem**

The proposed search strategy divides the search space into such hierarchical sub-spaces and uses a different search technique for each search sub-space in the hierarchy. The sub-space of machine configurations is generally small compared to the lower level search sub-spaces. Hence, a
local search technique, such as hill-climbing, may be used at this level of the search hierarchy. For searching through the sub-space of partitions for a machine configuration, the automation process may use one (or both) of two guided random search techniques: simulated annealing and genetic algorithms [5,6]. Likewise, for the search sub-space of mappings the automation process may use either or both of genetic algorithms or simulated annealing. The ability of guided random search techniques to find good solutions is determined by the objective function that is being optimized. Hence, depending on the objective function that is being used, the automation process could choose the search techniques to be used at levels S<sub>2</sub> and S<sub>3</sub>. Because this strategy allows for a mix of search techniques to be used in searching through the hierarchical search space, we call this strategy hierarchical hybrid search.

5. Related Research

As noted earlier, much of the existing research on the partitioning problem for applications implemented on parallel computers focuses on the back-end of the application development at which point software code for the application already exists. Existing research on partitioning can be categorized into two classes: research on data partitioning for data-parallel computations and research on functional partitioning [e.g., 3,7,8].

Much of the research in data partitioning comes from the work on compilers for data-parallel programs. The two key decisions that must be made in targeting a data parallel program to a multiprocessor architecture are data decomposition (e.g., block decomposition, cyclic decomposition) and data alignment (i.e., the size of the subarrays and the assignment of array elements to the decomposed subarrays) [7]. Languages such as High-Performance Fortran (HPF) [9] rely on the programmer to declare how arrays are to be aligned and to define the type of decomposition to be used. The language includes annotations such as BLOCK and CYCLIC for specifying the decomposition strategy. The compiler then generates the communication and synchronization code using necessary optimizations to reduce the resulting overhead.

A number of recent research projects have investigated strategies for automatic decomposition and alignment of data [e.g., 10-14]. The PARADIGM [10] compiler accepts Fortran 77 procedures as input and outputs a partitioning scheme for each array in the procedure. Kremer, et al. [12] discuss an interactive tool that assists a user in determining the data distribution. Chatterjee, et al. [13] describe a framework for automatic alignment of arrays in data-parallel languages. Mace shows that finding optimal decompositions is an NP-complete problem in the general case [15].

In the area of functional partitioning, Sarkar [16] discusses a framework for functional partitioning of a program for execution on shared-memory machines. The framework can be used for partitioning any program into parallel tasks provided that a program dependence graph (PDG) [17] can be computed for it.

For many applications, neither data parallelism nor functional parallelism is adequate alone, and both forms of parallelism must be exploited. Subhlok, et al. [18] describe the partitioning and mapping of a special class of applications that process a stream of input and can be expressed as pipelines of data-parallel computations. The PARADIGM compiler project [10] at the University of Illinois is addressing automatic partitioning of programs containing functional and data-parallel tasks.

In the area of software mapping, the existing research can be divided into three categories -- a) mapping of data-parallel computations by compilers, b) mapping of functional-parallel applications (expressed as task graphs) and c) mapping of functional-parallel pipelines of data-parallel computations. Mapping consists of assignment of program data, computations, and communication to the machine resources. In general, mapping is treated as a combinatorial optimization problem, and a wide variety of optimization algorithms have been applied.

In data-parallel computations, the mapping is driven by data distribution. HPF and Fortran D follow a two-stage model for partitioning and mapping data [7]. In the first stage, users specify the alignment of arrays to templates and the mapping of those templates to a user-defined rectilinear grid of logical processors. The second stage assigns the logical grid to physical processors. Once the data is mapped, computation may be mapped using the owner computes rule according to which the computation of values in assignment statements are assigned to the processor associated with the variable on the left hand side of the statement. Alternatively, all the work associated with a loop iteration may be assigned to the same processor.

These techniques were initially applied to problems where the data access patterns are regular. Agarwal, et al., address compilation and run-time techniques to map data in irregularly-coupled and multi-grid problems [19]. They analyze the array access patterns in the program and generate efficient interprocessor communication schedules. Ponnumasy, et al. address the mapping of irregular problems where the data access patterns are not known at compile time [20]. At run-time, data access patterns are examined, external partitioners are invoked to distribute
program data, and schedules are generated for efficient data communication.

Some of the early work in task assignment was done by Bokhari [21], who showed that the mapping problem is similar to complex combinatorial problems such as graph isomorphism and quadratic assignment problems. Since then, a variety of optimization algorithms have been used, including those based on graph theory, mathematical programming (e.g. dynamic programming), network flow, physical optimization (e.g. simulated annealing, neural networks), heuristic searches, and genetic algorithms [22].

Little work has been done in the area of mapping computations that exploit task parallelism as well as data parallelism. One exception is the work reported by Subhlok [18]. They use simple heuristics to map pipelines of data parallel computations to processors. They build an execution model of the application using timing information from trial executions, and use it to compute the best mapping.

The research efforts surveyed above primarily focus on the partitioning and mapping problem as it applies after the coding phase of the system development process. In contrast, our research focuses on the "front-end" system definition and design phase of the system development process when the application code may not yet be developed. Another distinguishing aspect of our proposed approach is the hierarchical hybrid search strategy that allows for the use of a combination of different search heuristics to search through the space of possible partitions and mappings. This approach for automated software partitioning and mapping has the potential to handle a broader range of applications and objective functions than is possible with the existing approaches described above.

6. Conclusions

Automated software partitioning and mapping is critical for the early evaluation of design choices during the system definition phase of the application development process for multiprocessors. We propose an approach for building front-end design tools that will automate the partitioning and mapping process and thereby facilitate rapid and cost-effective development of applications for distributed memory multiprocessors.

The innovative aspects of the proposed design of the partitioning and mapping tools include:

1. A new approach for representing the high-level design of the application software that facilitates automated partitioning and mapping. This representation called the Augmented Task Dependency Graph (ATDG) allows one to express functional parallelism as well as data parallelism at a high-level.

2. An approach for representing the characteristics of distributed memory multiprocessors that allows for automated search through the state space of architectural possibilities.

3. A new strategy for searching through the possible space of design choices, called hierarchical hybrid search. The strategy organizes the search space as a hierarchy of sub-spaces. Different techniques are used for searching through the different search sub-spaces. Techniques employed include local search using hill-climbing and global search methods based on simulated annealing and genetic algorithms.

This paper presents the preliminary results of an ongoing research project aimed at establishing the feasibility of the proposed framework for automated software partitioning and mapping for distributed multiprocessors. To fully establish the viability of the proposed approach, it is necessary to assess its effectiveness in producing high quality solutions for a variety of user-specified objective functions. Such assessment requires a prototyping effort. We propose to implement and evaluate a prototype of a partitioning and mapping toolset, based on the proposed approach, during the next phase of this research project.

References


