Abstract

The creation of system models and the development of parallel computer software models are important as a basis for the creation of any parallel computer application. They can provide a simplified view of complex systems when a high level of abstraction is used. Abstraction is a method of extracting the important details of a system and placing them into a formal model. The major types of models for parallel systems are based on the amount of control the designer has over the implementation of the system. The designer may have complete control or the tools may take care of all aspects of parallelism, decomposition, mapping, communication, and synchronization of tasks. Within the different kinds of models there are three sub-categories, which include, dynamic tasks creation with unlimited communication, fixed task creation with unlimited communication, and static task creation with limited communication.

Introduction

For an application to be efficiently designed and programmed, the designer must have some basis from which to understand the characteristics of the system. The purpose of the model is to provide a high level representation of the system to simplify the design and understanding of its behavior. Models provide various levels of abstraction that are used to simplify very complex applications. The level of abstraction dictates the amount of detail the application designer must be concerned about in the design. The higher the
level of abstraction the less need for attention to be focused on the details, which allows for faster application development.

In parallel software, the level of abstraction is driven by the choice of software development tools. Six different levels of abstraction are provided by parallel computing libraries:

1. When parallelism is abstracted out completely, the developer is unaware that the application will run concurrently. Therefore, compiler and run-time environment to decompose the application.

2. Parallelism can be specified in the application by the developer, but the developer has no control over the decomposition of the application. Hints are given to the tools via language constructs.

3. Parallelism and decomposition are handled by the developer, but mapping, communication, and synchronization are handled by the tools.

4. The developer has taken on the mapping of tasks to the appropriate processing elements, and must be aware of the performance of the interconnection network. Major performance loss could occur, if there is any amount of inter-task communication.

5. The amount and method of communication is determined by the developer.

6. The developer must decide every aspect of the application. However, using this level of abstraction can be difficult to design and implement, since every detail must be covered.
Level of abstraction relates directly to the ease of implementing the application.

To ensure the quality of the programming model, the gap should be examined between the amount of information provided by the application design and the amount of information actually needed to implement the application. A solid software development methodology must be used to prevent errors in the implementation of the application, because of the extreme difficulty in debugging a parallel application. When multiple tasks are spread across a system with multiple states, it is difficult to monitor the interactions.

Parallel programming models can also affect the portability of the application. If the design relies on any feature of the system hardware, it makes it difficult to port to other systems. A model should be independent of the hardware that the application runs on.

Software models represent an application in formal method, to ensure easy comprehension. If a problem arises in the system, a model can often be helpful to identify interactions that could potentially cause a problem. A parallel software model must simplify a system to a point where it is understandable by the designers and implementers.

High level abstraction that relies heavily on compiler tools and run-time support can cause efficiency and performance problems across numerous architectures. A designer must find a happy medium between performance and simplicity.
The final measure of a model is the cost, which is usually made up by performance, utilization of the processors, and complexity of development. This is where the designer must take care in selecting a programming model. Some of the measures just previously mentioned seem to lead to direct conflict with each other. An example could be selecting a programming model, which abstracts parallelism out completely. This could greatly affect system performance since usually compiler tools do not produce code that is not as efficient as hand-tuned code.

**Types of Models**

Within each model type 3 sub-categories can exist:

- *Dynamic thread allocation with unlimited communication* – This model supports runtime process creation and destruction with no limits on the amount of communication that can occur between the processes.

- *Static thread assignment with unlimited communication* – This type of model has a fixed number of processes but no limit on the amount of inter-process communication.

- *Static thread assignment with limited communication* – This model has a fixed number of processes with a fixed amount of inter-process communication.

1 Nothing Explicit

Nothing explicitly programmed is an ideal environment for developers, all of the decisions on parallelism are determined by the compiler and run-time tools. This is an ideal environment since the developer must not be concerned with any parallelism issues.
**Dynamic thread allocation and unlimited communication**

In Graph Reduction, functions are broken up into tree representation by the tools and then computed. A sub-tree is taken from the structure, computed and placed back into the original tree. After all sub-trees have been processed, the final result is the solution [1]. If there are independent sub-trees, they can be computed concurrently.

**Static thread assignment with unlimited communication**

Pre-defined blocks called skeletons are used to build a larger system. The skeletons hide the complexity of their functionality and can be concurrent in their implementation without the knowledge of the developer [3]. Since the skeletons are independently and internally concurrent, they can be strung together sequentially with no problems. Software tools should be able to map the threads to the processors.

**Static thread assignment with limited communication**

To limit communications homomorphic skeletons are constructed that operate on single data structure such as arrays or trees [3]. Since the size of the data structures are known and can be repeatedly used, the developer has some control and an ability to estimate the amount of task creation and communication. The internals of these skeletons can be implemented in a highly concurrent method since the data types they operate on are uniform. The skeleton developers can take advantage of this knowledge and realize the skeletons very efficiently.
2 Parallel Explicit

In parallel explicit model, the existence of parallelism is known by the developer and must be made clear in the program. The developer can take advantage of his global understanding of the application to exploit known concurrency.

*Dynamic Thread allocation with unlimited communication*

Dataflow involves simple computations that are expressed as operations with specific inputs and outputs. The order and time of execution depends solely on data dependencies. If no data dependencies exist between two operations, they can be executed concurrently. Each loop of a repetitive statement causes a new thread to be created. Since operations execute at different times, there is no advantage to temporal locality. Decomposition has little effect on performance of Dataflow [3]. The compiler controls communications by the naming of variables and there is no synchronous communications. Language is abstract and simple, but does not have a natural software development methodology [3].

*Static Thread allocation with limited communication*

Data parallelism of functions occurs when all elements of a data structure are processed in parallel, similar to the idea behind Single Instruction, Multiple Data (SIMD) machines. Certain languages such as High Performance Fortran (HPF) have language constructs that allow the developer to indicate that an operating can be handles in parallel [3]. However, this model can greatly limit the flexibility of the system, since operations are chosen based on their ability to be implemented efficiently. The programmer must be cautious about data referenced inside of these constructs and not reference identical memory locations [3].
3. Decomposition explicit

Explicit decomposition occurs when the designer is aware of parallelism, and in return must decompose the application tasks. This model has the ability to tune the processor loading to achieve a the most cost effective utilization of the system processors. Developers may be able to have a better understanding for the loading issues, which drive the decomposition of the work.

Static Thread allocation with limited communication

The Bulk Synchronous Process (BSP) model consists of p abstract threads each with its own local memory, with only two system properties that is of concern, the time it takes for barrier synchronization and what rate data can be delivered [3]. Those two system parameters are major factors in determining the systems performance. The BSP model operates a number of threads in a time frame called supersteps. A superstep consists of computation, global communications, and barrier synchronization [2]. A superstep starts when all threads start their computation phases. When all computation has ceased, the threads perform any global communication that needs to occur. Finally, a global synchronization is performed to ensure that all the threads have reached the same point.

4. Mapping explicit

The developer performs explicit mapping when he needs to control the allocation of tasks in the system. For example, a simple system has two processors, a floating point processor and a fixed-point processor. The application being developed on this system
has two tasks, one that is performing data encoding and the other is calculating phase error for a phase-locked loop. The data encoding task need logical bit manipulation capabilities while the phase error calculation requires high accuracy. Naturally the phase error task needs to run on the floating-point processor and the data encoder needs the logical operations of the fixed-point processor. The developer would be aware of the task requirements and could assign the tasks to the appropriate processor. This is a simple example, but should illustrate the requirement for the developer to control the mapping of tasks.

*Dynamic Thread allocation with unlimited communication*

Remote Procedure Call (RPC), may send out a message to a distant processing element for a procedure to take place. The caller, on the other hand, must block until the procedure returns with the required data. To overcome this inefficiency, a thread based RPC system would allow for other operations tasks to execute while waiting for the procedure to return [3].

*Static thread assignment with unlimited communication*

Graphical Languages models allow the programmer to insert communications into the application at a higher level. In the Enterprise language, the compiler classifies each program block into a type and automatically inserts communication structures based on these types [3].
5 Communications explicit

There will be systems that need to control the amount of communication that occurs, since communication can have a major affect on the performance of an application. An example could be a simple system where the processors are connected on a single bus. The developers would like to limit the amount of communications that occurs across the bus since any large amount of communication could lead to saturation of the bus. Saturation would severally degrade system performance.

*Dynamic thread assignment with unlimited communication*

Process Nets resemble the Dataflow model, in that they are made up of independent entities that respond to the arrival of data. Individual decisions are made on certain operations that will be performed on the arrived data. However, Process Nets unlike Dataflow lack a global state. These Actors receive incoming messages that are queued into a buffer and processed sequentially. To eliminate the occurrence of bottlenecks in the system, Actors are grouped together. If an Actor of a group is busy, another Actor will step in and handle the next incoming message [3].

*Static thread assignment with unlimited communication*

The Internal Object Oriented Languages model, includes Mentat Programming Language (MPL) was designed to address the need for architecture independent parallel application development [3]. The data driven aspect provides a high degree of parallelism while the object oriented design does an excellent job of hiding the synchronization issues. In the Mentat environment, object oriented encapsulation is used to hide the internal operations
of class member functions and data dependency issues from the developer. The compiler and run-time tools can identify data dependencies between instances of objects and inserts automatic synchronization and communication support [5]. This allows the developer to worry more about task decomposition.

*Static thread assignment with limited communication*

Systolic arrays use a fixed number of processing element that work in unison to perform repetitive computations [1]. These arrangements are called systolic because they perform their action in beat with each other, similar to the beating of a heart. The language is used to specify mapping of tasks, data flow through and operations of the processing elements [3].

6 Everything explicit

There will be system, especially heterogeneous networks of workstations (NOW), which need to have every aspect of the application controlled. NOWs might need a flexible system model if more advanced tools are not available for some of the individual systems.

*Dynamic thread assignment with unlimited communication*

Parallel Virtual Machine (PVM) allows the developer complete control over every aspect of the system. PVM is a message passing system, which provides procedures for all parallel system operations such as creation of threads and communication. From these
basic procedures the developer can implement synchronization methods and task mapping.

Summary

The current push is in the direction of tools that hide all of the issues of parallelism. This is necessary if general purpose computing is going to embrace parallel processing. There will be a need for general purpose parallel processing in the future, since the physical limitations of transistors is starting to curb single processor performance gains. High performance computation will still push into developer control into the lower level models to gain performance, but as our understanding of parallel computing systems and architectures will increase, they will hopefully start rising to a higher level of abstraction.