Concurrent Approach to Data Parallel Model using Java

Bala Dhandayuthapani Veerasamy

Abstract—Parallel programming models exist as an abstraction of hardware and memory architectures. There are several parallel programming models in commonly use; they are shared memory model, thread model, message passing model, data parallel model, hybrid model, Flynn’s models, embarrassingly parallel computations model, pipelined computations model. These models are not specific to a particular type of machine or memory architecture. This paper expresses the model program for concurrent approach to data parallel model through java programming.

Keywords—Concurrent, Data Parallel, JDK, Parallel, Thread

I. INTRODUCTION

The term processor, or microprocessor, refers to the central processing unit (CPU) [1]. It’s a single chip responsible for the execution of instructions given by a computer program, memory management and address translation, integer and floating point operations, and cache management. Uniprocessors have a single processor. Application instructions and hardware calls are executed in order, one at a time (sequentially). The system appears to run concurrent processes, but the processor actually switches back and forth between instructions.

Two events are said to be concurrent if they occur within the same time interval. Two or more tasks executing over the same time interval are said to execute concurrently. Tasks that exist at the same time and perform in the same time period are concurrent. Concurrent tasks [1] can execute in a single or multiprocessing environment. In a single processing environment, concurrent tasks exist at the same time and execute within the same time period by context switching. In a multiprocessor environment, if enough processors are free, concurrent tasks may execute at the same instant over the same time period. The determining factor for what makes an acceptable time period for concurrency is relative to the application.

Concurrency techniques [3], [6] are used to allow a computer program to do more work over the same time period or time interval. Rather than designing the program to do one task at a time, the program is broken down in such a way that some of the tasks can be executed concurrently. In some situations, doing more work over the same time period is not the goal. Rather, simplifying the programming solution is the goal. Sometimes it makes more sense to think of the solution to the problem as a set of concurrently executed tasks. This technique is used in the parallel computer architectures.

Parallel programming and distributed programming [1] are two basic approaches for achieving concurrency with a piece of software. They are two different programming paradigms that sometimes intersect. In the past programming life, we were mostly using sequential programming. But, today’s life style is going with more faster than the past decades. Also, solving problems on the computers are enormous. Parallel computer [1] can executes two or more job within a same period of time.

Java is just a computer language [5] that has secure, portable, object-oriented, multithreaded [3], [4], [6], interpreted, byte-coded, garbage-collected, language with a strongly typed exception-handling mechanism for writing distributed programs [4]. Java is an object-oriented programming language, which added the new features such as overriding, interface and etc. Java supports multithreaded programming, which allows you to do many things simultaneously on the same time interval. Java enables the creation of cross-platform programs by compiling into an intermediate representation called java byte code. JVM (Java Virtual Machine) is an interpreter for java. Java is designed for the distributed environment on the Internet. Java has technology called RMI (Remote Method Invocation) that brings unparalleled level of abstraction to client / server programming. Byte code is a highly optimized set of instructions designed to be executed by the java run-time system, which is called Java Virtual Machine (JVM). Java handles de-allocation for you automatically, this technique called garbage collection. The Java Developers Kit (JDK) is a set of command-line tools that can be used to create Java programs. The current release of the JDK is version 1.6

II. THE CONCEPT OF DATA PARALLEL MODEL

The data parallel model [7] has characteristics that most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.

The tasks can perform the same operation on their partition of work. On the shared memory architectures, all tasks may have access to the data structure through global memory. On the distributed memory architectures the data structure is split...
A thread is in the "new" state when it is first created until its
start method is called. New threads are already initialized and
ready to get to work, but they haven't been given the cue to
take off and get busy. When the start method is called on a
new thread, the run method is in turn called and the thread
enters the "runnable" state. You may be thinking this state
should just be called "running," because the execution of the
run method means a thread is running. However, you have to
take into consideration the whole priority issue of threads
having to potentially share a single CPU. Even though every
thread may be running from an end-user perspective, in
actuality all but the one currently accessing the CPU are in a
"runnable" wait state at any particular instant. You can still
conceptually think of the "runnable" state as the "running"
state; just remember that all threads have to share system
resources. The "not running" state applies to all threads that
are temporarily halted for some reason. When a thread is in
this state, it is still available for use and is capable of re-
entering the "runnable" state at some point. For each of these
actions causing a thread to enter the "not running" state, there
is an equivalent response to get the thread running again. A
thread enters the "dead" state when it is no longer needed.
Dead threads cannot be revived and executed again. A thread
can enter the "dead" state through one of two approaches,
which the run method finishes executing or stop method is
called. The first approach is the natural way for a thread to
die; you can think of a thread dying when its run method
finishes executing as death by natural causes. In contrast to
this is a thread dying by way of the stop method; calling the
stop method kills a thread in an asynchronous fashion. Thread
class provides constructors, methods to use this concept.

The following program 1 represents the concurrent
approach to data parallel model. The result of the program is
given in Fig. 3. In the program, class DP consists of four
threads called “task1, task2, task2, and task4”. The DP class
consists of an array, initiated with totally 20 elements. In this,
the “task1” uses 0 to 4, “task2” uses 5 to 9, “task3” uses 10 to
14 and the “task4” uses 15 to 19. This is illustrated in the Fig.
2. All the four tasks or threads will concurrently utilize the array called data parallel programming model.

```java
//DP.java
class DP implements Runnable{
  int sum=0;
  Thread t;
  int aa[] = new int[20];
  String name;
  int beg, end;
  static int ln=0;
  DP(String str, int st,int en){
    t=new Thread(this,str);
    beg=st;
    end=en;
    name=str;
    t.start(); }
  public void run(){
    try{
      for(int i=beg; i<end;i++) {
        aa[i]=I; sum=aa[i];
        System.out.println("Thread "+name +": " +sum);
        ln =ln+1;
        if(ln==4){
          System.out.println("n");ln=0;}
        t.sleep(200);
      }
    }catch(Exception e){} }
  public static void main(String BDP[]){
    DP b1=new DP("task1",0,5);
    DP b2=new DP("task2",5,10);
    DP b3=new DP("task3",10,15);
    DP b4=new DP("task4",15,20);
  }
```