SWE 212 Analysis of Programming Languages

Week 6 Summary of 'Linda and Friends' by Sudhir Ahuja, Nicholas Carriero and David Gelernter

Samyak Jhaveri

'Linda and Friends' is an offbeat paper that describes in detail the details of a new computing model designed from the ground up to support parallel programming. It functions as an "add-on" to existing languages like Fortran or C to extend it and therefore enable it to create computational activities supported by parallel communication among those activities. The main motivating factor behind the need for a model like Linda is the simplification of concurrent/parallel programming, since it might be overwhelming to think and program in the context of multiple simultaneous processes running from a single piece of code. A Linda system has two components: a runtime kernel (to implement the inter-process communication and process management) and a preprocessor or compiler. Additionally, the kernel is language agnostic - meaning it can bless any language with its parallel processing features.

The Linda system operates on the concept of distributed data-structure methodology for implementing parallelism. This is different from most of the parallelization schemes of that time as parallelization was thought to be just a partitioning of the processes into many activities. However, Linda can transpire parallelism by both partitioning and replicating processes. This ability to replicate copies of processes rather than just generating new processes is the key ingredient in Linda.

The needs of parallel programmers

1. **A machine-independent and portable programming vehicle**. During the time of this paper's publication, there were many parallel machines popping up with their own architecture for parallelism but unfortunately, they lacked high-level machine independent tools. While some machines did have the essential system calls that provided a higher level of abstraction when using parallelism, they were not exhaustive for the myriad of ways people had started using parallel program designs. This meant that parallel programs written on one machine could not run on another machine with different parallel architectures, unless it was not just reconstructed but conceptually reformulated for a different machine.
2. A programming ool that relieves the user from being concerned about the temporal and spatial relationships among the parallel processes. Uncoupling a process into parallel/simultaneous activities requires management of the temporal and spatial resources of the CPU(s). Spatially, a process should be able to operate without the knowledge of the presence or existence of another process such that wherever input it receives and data it produces by performing its tasks should not be directed at another process but rather by "published" to the general memory space of the program for any other process to pick it up from there. This would make it easier to write parallel programs as it would relieve him/her of thinking about the flow of control and data simultaneously for all the parallel processes. Temporally, if a process is explicitly programmed to send a piece of data to another process, its functioning is constrained to

load both the processes simultaneously. It would be better if the Linda model was free of such scheduling issues.

3. Dynamic distribution of tasks. Although many systems have the feature of statically distributing the tasks to run in parallel, it is not very practical for scaling up the same program or for certain specific programs. In some situations, it is better to have a dynamic distribution of threads, especially as the complexity of the program increases.
4. A programming tool that can be implemented efficiently on existing hardware.There has to be a way to elegantly implement parallel language designs on any kind of hardware that supports parallel computing.

Next, the authors take the reader on a tour of Linda's inner functioning. Comparing Linda's idiosyncratic memory model with a conventional one, they point out some key differences:

Conventional Memory Model	Linda's Memory Model
Storage unit is a physical byte	Storage unit is a logical tuple (an ordered set of values)
Elements of the memory are accessed by addresses	Elements of logical tuples are accessed by logical name.
Accessed via two operations - read and write	Accessed via three operations - read, add, and remove

As Linda stores its data in tuples, it is possible to manipulate the data in-situ. This same trait allows parallel processes that use Linda to have a shared yet distributed data structure to access and perform tasks with the data. Linda's shared memory is called tuple space (TS). So, for processes to access the shared data, add it to and access it from the shared tuple space without ever directly contacting one another. There are four main operations that are used with the tuple space:

- **out()** - adds tuple t to the TS
- **in()** - withdraws tuple t form TS that matches with template s
- **read()** - the same as in() but the tuple t remains in the TS, and
- **eval()** - is the same as out() but that it adds an unevaluated tuple to TS.

Programming in Linda

Ideally, as stated above, a distributed data structure model complements the parallel programming model perfectly as it allows processes to interact with the data without interacting with each other. However, since it is impossible to implement this distributed data structure model in most parallel programming languages, the manager process model is a better solution. This model involves a manager process that carries out the manipulations on the data encapsulated within it as per the requests of the other processes. The authors claim that programming in Linda is fundamentally different from programming

in more common parallel programming languages with this manager processes model. Bringing back the theme of replicating processes vs. partitioning processes from an earlier part of the paper, the authors suggest that there it is intuitive to think that partitioned-network processes are coupled while replicated processes are uncoupled and that the workers in the latter model ignore other coworkers completely. Some favorable traits of replicated process model are :

1. Transparent scalability. A program with ten or a hundred parallel workers the same way as a single process program i.e. the program is easily scalable and the resources available to the program can be adjusted as per the number of workers/processes it has.
2. No need for context switching. As each processor runs a single process, the burden to manage the processes with context switching doesn't change with the presence of more than one node.
3. Dynamic load balancing. The processes keep looking for tasks to do during runtime.

The authors have referred to Linda as a programming language but they argue that it isn't.Rather, it is a new machine model, in the same sense in which data flow or graph-reduction may be regarded as machine models as much as programming methodologies.

Concluding the paper, the authors mention that Linda can be both an interpreted language as well as compiled one. Their pie dream would be to construct a Linda machine optimized for Linda primitives.