Need for Conversion of a Sequential Code to Parallel

Ambuj Kumar Agarwal [1], Dr Vinodini Katiyar[2]
[1]Department of Computer Application, Teerthanker Maheveer University, Moradabad, UP, India
[2]Professor & HOD, SRM University, Lucknow, UP, India
[1]ambuj4u@gmail.com

Abstract - **With the availability of multiple processing elements (PE) on a platform, high throughput can be gained by executing parts of an application concurrently on each PE or subset of PEs. An example of such a platform is a multiprocessor system on chip (MPSoC) where each processing element has its local memory as well as a common shared memory which all of them share. All PEs can run independently in a decentralized fashion or can run as a centralized model where one processor acts as a master. An application specified using an imperative model of computation can run only on a single processor and is not suited for multi-processor platform. The application needs to be modified to make it suitable for running on multiprocessor platform by identification and extraction of parallel tasks from the sequential code and efficiently mapping these tasks onto a multiprocessor platform. Manual identification and extraction of parallel tasks from sequential code is very time consuming and error prone as it requires complete analysis of sequential source code which is complex if it spans thousands of lines.**

Keywords- **Compaan, KPN, Sprint, Harmonic**

I. INTRODUCTION

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling.[2] As power consumption (and consequently heat generation) by computers has become a concern in recent years,[3] parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.[4]

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks.

Parallel computer programs are more difficult to write than sequential ones,[4] because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance.

Parallel computing, on the other hand, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

In this paper I have tried to emprise on the need for conversion of sequential code to parallel and comparison of existing automation tools.

Traditionally, software has been written for serial computation:

To be run on a single computer having a single Central Processing Unit (CPU);

A problem is broken into a discrete series of instructions.

Instructions are executed one after another.

Only one instruction may execute at any moment in time.

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem:

1. To be run using multiple CPUs
2. A problem is broken into discrete parts that can be solved concurrently
3. Each part is further broken down to a series of instructions
4. Instructions from each part execute simultaneously on different CPUs

Benefits of parallel computing on networks of PCs

Save time and/or money: In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. Parallel clusters can be built from cheap, commodity components.
Provide concurrency: A single compute resource can only do one thing at a time. Multiple computing resources can be doing many things simultaneously
Use of non-local resources: Using compute resources on a wide area network, or even the Internet when local compute resources are scarce.
Limits to serial computing: Both physical and practical reasons pose significant constraints to simply building ever faster serial computers:

Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30

cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.

Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.

Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

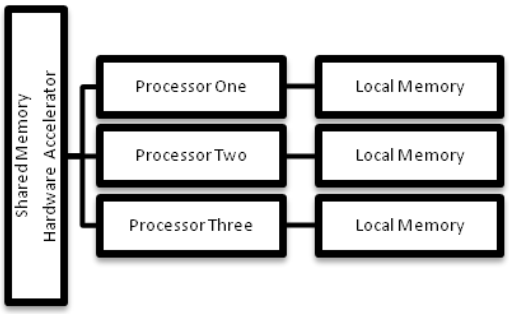

Figure 1 A Multiprocessor Soc Architecture Diagram

II. IDENTIFICATION OF THE PROBLEM

The problem of transformation of sequential application for mapping on multiple processors is not new and has been studied by many researchers in different contexts:

Selecting model of computation for parallel execution.

Automation for transformation from sequential model of computation to parallel model of computation.

Efficient scheduling of parallel model of computation on multiprocessor platform.

A. Selecting model of computation for parallel execution

In parallel models of computation, data is communicated between N processors in two ways:

1. Shared memory
2. Message passing

In shared memory communication each processor has access to shared memory. If a processor updates the value of the variable stored in shared memory then the other processor can read the value of that variable from shared memory. All processors can gain access to shared memory simultaneously if memory locations they are trying to read from or write to are different.

If one processor is good, N processors are GREAT:

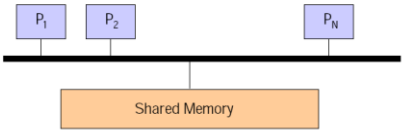

Figure 2 A Shared Memory

IDEA:

1. Run N processes, each on its OWN processor!
2. Processors compete for bus mastership, memory access
3. Bus SERIALIZES memory operations (via arbitration for mastership)

In Message passing communication each processor has only local memory. Processors communicate by sending and receiving messages in the form of data. The communication can be point-to-point where data is only transferred between exactly two processors or it can be broadcast where a processor sends data to all other processors. Programming a message-passing multicomputer can be achieved by

1. Designing a special parallel programming language.
2. Extending the syntax/reserved words of an existing sequential high-level language to handle message passing.
3. Using an existing sequential high-level language and providing a library of external procedures for message passing.

Here, we will concentrate upon the third option.

Necessary to say explicitly what processes are to be executed, when to pass messages between concurrent processes, and what to pass in the messages.

Two primary methods are needed in this form of a message-passing system:

1. A method of creating separate processes for execution on different computers

2. A method of sending and receiving messages

I. Single Program Multiple Data (SPMD) model

Different processes are merged into one program.

Within the program are control statements that will customize the code; i.e. select different parts for each process in figure 3

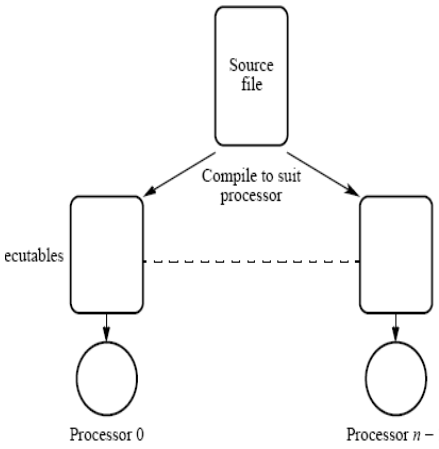

Figure 3 Passing a message between processes using send() and recv() library calls.

II. Synchronous Message Passing

Routines that actually return when the message transfer has been completed.

Do not need message buffer storage. A synchronous send routine could wait until the complete message can be accepted by the receiving process before sending the message. A synchronous receive routine will wait until the message it is expecting arrives. Synchronous routines intrinsically perform two actions: They transfer data and they synchronize processes. Suggest some form of signaling, such as a three-way protocol in figure 4.

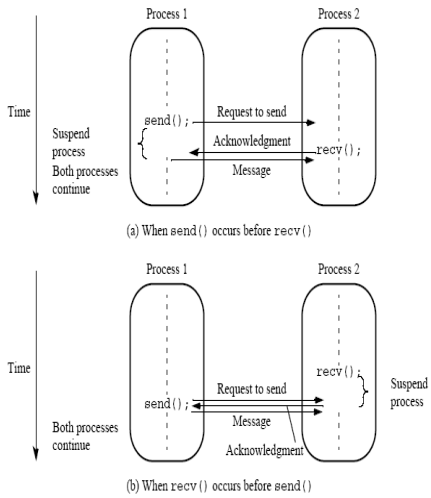

Figure 4 Synchronous send () and recv() library calls using a three-way protocol.

Generally, a message buffer is needed between the source and destination to hold message. Message buffer is used to hold messages being sent prior to being accepted by recv().

For a receive routine, the message has to have been received if we want the message. If recv() is reached before send(), the message buffer will be empty and recv() waits for the message. For a send routine, once the local actions have been completed and the message is safely on its way, the process can continue with subsequent work. In this way, using such send routines can decrease the overall execution time as in figure 5.

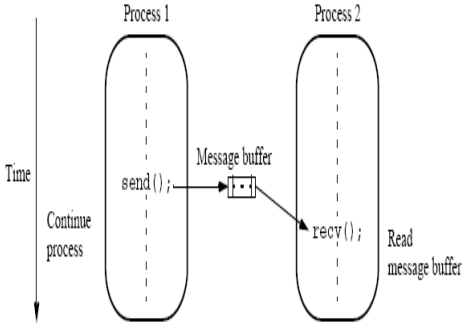

Figure 5 Using a message buffer.

Efficient mapping of an application onto multiprocessor platforms could be achieved if applications are specified as parallel model of computation. Several parallel models of computation exist [3], but I found that two major models of computation namely -Dataflow Process Network [2] and Kahn Process network(KPN) [9] are convenient to specify streaming applications such as multimedia and signal processing. Both Kahn process network and dataflow process network use point-to-point message passing type of communication. A process network shown in Figure 6 consists of concurrent processes that are interconnected by directed first-in-first-out (FIFO) channels which are called fifos. In Figure 6 processes are represented by circles. Each process has an input and output port through which it communicates with its environment. Each port is connected to precisely one fifo and each fifo is connected to exactly one output port and exactly one input port. A process performs a sequence of computation and communication actions that may be interleaved.

AKahn Process Network is a process network where size of a fifo channel is unbounded. Processes produce tokens (data elements) through their write interface that are sent along communication channels and consumed by destination processes through their read interface. Reads from the channel are blocking and writes are non-blocking.

A Dataflow process networks are a special case of Kahn Process Networks. Here each process is an atomic actor called dataflow actor. Instead of using the blocking read semantics of KPN, dataflow actors have firing rules. These firing rules specify what tokens must be available at the inputs for the actor to fire. When an actor fires, it consumes some finite number of input tokens and produces some finite number of output tokens.

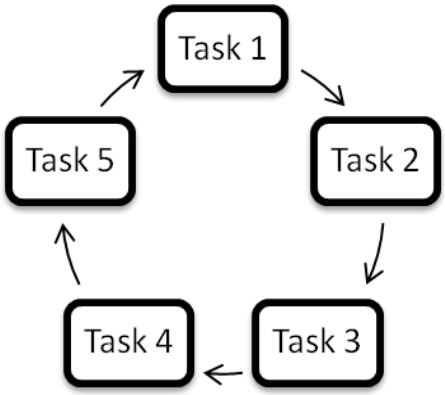

Figure 6 A Process network

Another variant of process network is a task graph which is a special case of dataflow process networks.A task graph is a network of tasks connected by communication channels. In a task graph, computation and communication is not interleaved as is the case with KPN. Also, data must be available on all input ports of the task before it starts executing.

B. Efficient scheduling of parallel model of computation on multiprocessor platform

In this area, scheduling of parallel model of computation on multiprocessor platform is studied to ensure correct execution of the model with increase in throughput. Much of this work belongs to scheduling of process networks on multiprocessor platforms. Detection and resolution of "artificial deadlocks" in process networks has been discussed in [2] [6] [10]. A process network is said to be "artificially deadlocked" if a chain of blocked processes exists in process network as a result of fifo channel being full.

Run-time scheduler [8] [10] [11] have been built that execute process networks on multi-core platform to test functionality and effectiveness of generated process network in terms of throughput.

III. COMPARATIVE ANALYSIS OF AUTOMATION TOOL

Existing tools such as Compaan [2], Sprint [6] and Harmonic [10] automate such transformation but each has its own characteristics, advantages and drawbacks. We have emphasized on above 3 tools and compared them as they are representative of the major work done in automating the transformation of sequential application for mapping on multiprocessor platform. Here we discuss details and characteristics of each tool and compare them on various parameters.

Compaan: The Compaan tool converts a sequential Matlab application into process network in a systematic automatic way. It converts a Matlab application into a polyhedral reduced dependence graph which is a compact representation of a dependence graph(DG) using parameterized polyhedral. Compaan works on partitioning of loops into tasks. It is confined to operate on affine nested loop programs (ANLP).

Sprint: Converts a sequential C application to executable concurrent model in System C. It facilitates generation of KPN as per designer specification by automatically detecting and inserting required communication channel in accordance with designer-specified task boundaries. Sprint only performs function level partitioning and does not handle loops.

Harmonic: It receives a C application as an input and generates an executable image for reference general purpose processor. It performs both functional and data level partitioning. It treats each function as a separate task. Harmonic does not work on KPN model of computation as it does not include channel generation step. Task partitioning and mapping tasks to general purpose processors (GPP) are the two phases implemented by Harmonic.

In the following table we have compared [3] approaches to automated transformation on these parameters:

1. Input application type: Specifies the language in which input application is written.
2. Final output type: Specifies the output form of the transformed application.
3. Type of parallelism handled: Specifies if type of parallelism is functional or data.
4. Transformation based on KPN: Specifies if the transformation generates a KPN model of computation.
5. Extent of automation: Specifies if the process of transformation is completely automatic (requiring no user decision).

IV. CONCLUSION

Now a days, there are many multiple processing units are available on a particular platform. For concurrent execution of the multiple applications, some automation tools are available in the market such as compaan, sprint, and harmonic. But they work according to the some specific constraints, for example compaan tool converts a sequential matlab application into process network. Sprint converts a sequential application to executable concurrent mode in system C and Harmonic receive a C application as an input and generate an executable image. So there are many limitations with the automation tools. For further processing, we need more automation tools to execute the application concurrently on multiprocessor platform. In which we require efficiently mapping to avoid manual identification and extraction of parallel tasks from sequential code and make it error free because it has spans thousand of lines of source code.

V. REFERENCES

[1] Atomium, http://www.imec.be/atomium.

[2] B.Kienhauis, E., and E.F.Deprettere. Compaan : Driving process networks from Matlab for embedded signal processsig architecture. In proceedings of Eighth International Workshop CODES (2000).

[3] B. Vanhoof,M. Peon, G. Lafruit, J. Bormans,M. Engels, and I.Bolsens, "A scalable architecture for MPEG-4 embedded zero tree coding," in Proceedings of the 21st IEEE Annual Custom Integrated Circuits Conference (CICC '99), pp. 65–68, San Diego, Calif, USA, May 1999.

[4] Dave, B., Lakshminarayana, G., and Jha, N. Cosyn: Hardware-software co-synthesis of heterogeneous distributed embedded systems.Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 7,1 (Mar 1999), 92-104.

[5] F. Catthoor, E. de Greef, and S. Suytack, CustomMemoryManagement Methodology, Kluwer Academic Publishers, Boston, Mass, USA, 1998.

[6] J. Cockx, K.Donolf, B., and R.Stahi. SPRINT:A tool to generate concurrent transaction-level models from sequential code. In EURASIP Journal on Applied Signal Processing (January 2007), vol. 1, p. 213.

[7] K. Denolf, et al., "A systematic design of an MPEG-4 video encoder and decoder for FPGAs," in Proceedings of the Global Signal Processing Expo and Conference (GSPx '04), Santa Clara, Calif, USA, September 2004.

[8] K. Denolf, C. De Vleeschouwer, R. Turney, G. Lafruit, and J.Bormans, "Memory centric design of an MPEG-4 video encoder," IEEE Transactions on Circuits and Systems for Video Technology, vol. 15, no. 5, pp. 609–619, 2005.

[9] P.Held. Functional design of data-ow networks. PhD thesis, Delft University of Technology, 2009.

[10] Verdoolaege, S., Nikolov, H., and Stefanov, T. pn: a tool for improved derivation of process networks. EURASIP J. Embedded Syst. 2007, 1 (2007), 19-19.

[11] W.Luk, J.G.F coutinho, T., Y.M.Lam, W., and K.W.Susanto,O.Liu, W. A high level compilation toolchain for hetrogeneous systems. In IEEE international SOC conference (Sept 9-11 2009).

[12] http://cgi.csc.liv.ac.uk/~igor/COMP308/ppt/Lect_21.pdf