**Compiler infrastructure to support power aware course grained reconfiguration in FPGA’s**

Main goal of the research – To provide compiler suite to allow for compile time application scheduling support for dynamically reconfigurable architectures. Large part of research will involve exploring topics along lines of adaptive application scheduling/architecture reconfiguration based upon power/performance characterization of application benchmarks. The compiler will exploit the following –

Premise of our research is that by dynamically programming the board memory on the fly using fast on-chip configurable memory (MRAM based), we should be able to achieve substantial savings in terms of power in course grained reconfigurable architectures without having any substantial degradation in quality of service of application benchmark runs. To support this, the compiler will need to make decisions as to how the code will be scheduled runtime for achieving optimal power/performance. At a very high level, the compiler will be helpful to configure number of cores the applications are running on and the network topology by fast reconfiguration of the architecture using the new memory architectures which provide support for fast reconfigurability. In addition, the compiler will also make decisions on code restructuring for optimal execution on course grained reconfigurable architectures.

**Section 1 - A very high level view of the compiler architecture -**

At a very high level the compiler will do the following –

**Phase 1 - Front end -** The front end will be a StreamIT based infrastructure which can take an algorithm expressed in form of a parallel program with structures for fork/join/round robin scheduling etc. It has been demonstrated that STREAMIT compiler can successfully describe a variety of signal processing algorithms. With this front end interface, backend infrastructure will be developed that takes this description and does compiler optimizations for optimal code generation.

Background research supporting this idea - There is lot of research done with STREAMIT compiler to demonstrate how to describe complex signal processing applications using this infrastructure. I played around a bit with the compiler and it appears like it is not too cumbersome to handle intermediate stages of this compiler and the structure is a very layered one. The individual parts of the compiler are very well documented and so modifying algorithms which handle code generation or optimizations is easy to integrate with this compiler interface.

**Phase 2 - Phase characterization of power signature of benchmark –** This task primarily involves splitting the application into phases of execution based upon power analysis. In this phase, we will need to develop algorithms to perform phase detection of benchmarks based upon power signature of benchmark execution. These algorithms may use some trace based execution hints, profile guided hints or other metrics (maybe pure basic block based characterization of application). The end result will be that the compiler will split the application into phases based upon power consumption of the application execution. Somehow, at this level, the communication aspects of the application will also have to be taken into account to study how to efficiently parallelize the code to execute on individual reconfigurable cores. The output of this phase should be guiding the phase 3, where compiler reorders code and provides hints to reconfigure hardware dynamically.

Background research supporting the idea – There has been a lot of research in terms of characterizing phases of execution of benchmarks. This work however has been mainly focused on performance characterization. There needs to be substantial research that needs to be done to characterize application phases for power signatures. A large part of my research will focus on generating these phases accurately and validate them. In addition, we have to quantify the power consumption in scenarios where there is communication going on between various processing elements. Till date, research in this area has been very limited and actually only one paper has barely even touched this aspect in context of chip multiprocessors, that too quantifying only performance aspects [1]. So quantifying the power analysis phases along with communication and memory architecture specific to course grained reconfigurable architectures will be a substantial challenge for which we need to develop algorithms and validate them.

**Phase 3 - Take the compiler hints generated in phase 2 and use them to reconfigure hardware/modify application for reprogramming the MRAMs dynamically** – By the end of Phase 2, we would have generated the necessary hints for code regeneration for RCGAs. We will need infrastructure to generate target architecture specific code based on the hints. A large part of this work will be in area of code generation phase of the compiler for retargetable architectures which is quiet challenging. In addition, this is the phase where we will also decide how to change the hardware architecture based upon hints provided, for example the decisions such as how many cores are needed to execute a phase of an application will be decided in this phase and the hardware architecture reconfigured accordingly.

Background research supporting the idea – There is some research done in this area where identification of performance phases has been used to dynamically modify memory hierarchy in general purpose processors [2] or dynamically changing microarchitectural features like cache and pipeline sizes [3]. There are a few more papers addressing other micro architectural features too. In addition, there are papers that talk about how to do compiler optimizations [4,5]. However, all these efforts primarily focus on phase classification based upon architectural performance metrics as opposed to power metrics. Our research will primarily focus on supporting compiler optimization and microarchitectural feature modifications (in our case RCGAs) based on power classification of benchmark execution.

Figure 1 represents a rough outline of this research effort –



Figure 1: Compiler framework for course grained reconfigurable architectures

**References -**

[1] Zhenlin Wang, Kathryn S. McKinley, Doug Burger. Combining Cooperative Software/Hardware Prefetching and Cache Replacment

[2] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In 33rd International Symposium on Microarchitecture, pages 245–257, 2000.

[3] A. Dhodapkar and J. E. Smith. Dynamic microarchitecture adaptation via co-designed virtual machines. In International Solid State Circuits Conference, February 2002

[4] R. D. Barnes, E. M. Nystrom, M. C. Merten, and W. W. Hwu. Vacuum packing: Extracting hardware-detected program phases for post-link optimization.

[5] M. Merten, A. Trick, R. Barnes, E. Nystrom, C. George, J. Gyllenhaal, and Wen mei W. Hwu. An architectural framework for run-time optimization. IEEE Transactions on Computers, 50(6):567–589, June 2001.

**Section 2 -**

Preliminary steps for understanding the course grained reconfigurable architectures and how to make decisions on such optimizations –

Steps involved –

1. **Understand how to use existing reconfigurable architecture board on which we are running applications and performance of those applications -**

Procedure –

1. I haven’t yet gotten to the board level aspects, but other than that I have gone ahead and touched everything else necessary to understand the current tool setup aspects. Over last week, I read a few more papers on dynamically reconfigurable architectures. In addition, I was able to synthesize and place and route a nios based system with multiple cores. I was also able to get a few simulations going with modellsim. I now have the complete tool setup and am able to do my own experiments. I might still need a quick tutorial on programming the board at some point, but it is far less critical right now as Xiaobin is busy with his proposal and I have other things to close on. My next task here is to actually generate execution input sequences based upon algorithms into simulation environment and get simulations going with various cores in NIOS systems.
2. I worked a bit on SPREE and am now able to generate SPREE cores based upon configurations (most of the ones I generated were from manual). I will try to place and route these IP’s and then the next step will be to integrate them and replace the NIOS CPUs for SPREE CPUs. There is a large amount of work here in terms of interfaces needed to talk to the other IP in the system when using a generic core like SPREE instead of NIOS cpu
3. Studying the STREAMIT compiler – I have downloaded the source-code, gone through the manual and started playing around with writing code for STREAMIT applications. The first step here will be to implement the algorithms which Xiaobin has implemented using STREAMIT compiler. That way, we can have benchmarks that we can parallelize and automatically generate application code as opposed to doing it manually the way it is done today.
4. Start understanding the switchbox architectures and outline the architecture of switchbox that I have to develop to help Xiaomin – I have started studying this aspect. I now have done quite a bit of background research on type of switchboxes that need to be implemented. The three requirements I have identified are low power, fast reconfigurability support and good performance and am studying how to implement such a switchbox. Another requirement will be integration of the switchbox seamlessly into the existing NIOS architecture. So we will need to provide interfaces to support the integration. I am looking towards all these aspects. Sometime by end of next week, I want to analyze power/performance of a few of the architectures I have identified and will compare them in form of a technical report and send it out.
5. **Understand the runtime memory reconfiguration and compiler infrastructure needed to support such reconfigurability**

Procedure –

1. Study the MRAM based architectures and integrate them into existing simulation environment
2. Develop interfaces to talk between Memory module, switchbox and processor (maybe SPREE or NIOS)
3. Characterize the MRAM memory with correct technology (delay, response aspects, power dissipation etc should be accurately represented for the fast memory to derive the power/performance metrics)
4. Enhance the compiler to reprogram MRAM to do dynamic reconfigurability of both cores and network architecture
5. **Start investigating compiler specific optimizations/scheduling constraints especially wrt application specific optimizations for power**

Procedure – I have outlined some specifics of this task in Section 1