CSC 400 : Problem Seminar Course

Proposal for a project on

Unified Heterogeneous Specialized Device Architecture (UHSDA)

1. **Introduction**

There is a tremendous increase in focus on GPUs for compute heavy applications in the recent past. The research in CPUs is currently augmented but the GPU market so as to supplement and produce high performance machines. What initially started as vector processors for graphics applications, translated into multicore SIMD (Single instruction multiple data) processors. They are so good at exploiting parallelism but have limitations of their own. It is for this reason that GPUs are not currently the host processors but mostly used as co-processors or accelerators.

The moment the potential of GPUs were shown, a lot of changes started taking place in the computer architecture and the high performance computing community. Adapting GPUs to other algorithms apart graphics is being looked at and various high speed libraries, interconnects between processors and GPUs are being developed and with all these, GPUs are also becoming more intelligent and more equipped to handle various types of offloaded computations from the general purpose processors. More effort is currently taken up by CPU developers and manufacturers to accommodate GPUs in the same die rather than an extra chip. As a result of all these changes, the compute power of the everyday computer and the best super computers in the world are being multiplied.

The aim of this proposal is to extend the integration beyond GPUs and to consider FPGAs as the other alternatives for co-processing. FPGAs are field programmable gate arrays. They are hardware that could be configured in real time into any hardware functionality that the users deems fit. Also, because it is real hardware it is extremely precise in terms of run times and accounts for every clock. They generally need a processor to configure them and they can be dynanamically reconfigured on the fly to adapt a different functionality. Though they are very verstile devices, similar to GPUs, they cannot be programmed with C or any normal language. FPGAs use some sort of a hardware description language like how GPUs use extensions so support the SIMD model. We will target identifying weaknesses of the current CPU/GPU approach and the various bottlenecks that it can present and how the introduction of an FPGA can ease those problems and how the compute power has a potential for drastically improvements. For the analysis, we will use state of the art FPGAs, GPUs and CPUs and model various interconnects and gather information about the nature of the applications that will be benefitted and the type of extra work that will be necessary with the introduction of such a compute platform. Essentially, multithreading of CPUs is much different from the multithreading of the GPUs. CPUs perform well when the thread is largely independent of other threads and when the threads have significant computations and independent code from other threads that are involved. There is no limitation on the type of computation that a thread can do or cannot do in case of CPUs. This is not the same when we consider a GPU. GPUs are good at handling non-diverging threads; suffer a high penalty

on any dependency on other threads or high memory bandwidths due to smaller caches per core and thread. Scheduling for GPUs are also simpler compared to the scheduling on CPUs. GPUs give a large throughput where there is enough parallelism to exploit.

2. Motivation

It is in these circumstances FPGAs come out to be very useful. FPGAs can be used for performing tasks that are varying in size and all in parallel with one another. GPUs need the same thread to run on all their cores and rely on data parallelism. It cannot exploit parallelism in a case of presence of heterogeneous threads. CPUs can take advantage of heterogeneous threads but are limited to the lesser number of cores that they have. FPGAs can run all such heterogeneous programs independently without the intervention of the other processors. They also have the advantage of having embedded RAMs and dedicated resources that all the processes that run on the FPGA are truly parallel. There are various types of applications for FPGAs in the present and just like GPUs, they have a huge potential that is waiting to be tapped into.

FPGAs are the platform of choice for a variety of applications where high performance and a number of concurrent tasks are needed to be executed on the same platform. Examples include embedded applications, high stream video encoders, signal processing stations, etc. Their applications are currently limited largely due to cost and specialization necessary to develop dedicated algorithms and mechanism. This is always the case before a technology is incorporated into the general stream. Once it is made a part of the normal processing chain, the necessary support will arrive.

There is one main disadvantage of GPUs that need to be highlighted. Though GPUs have hundreds of cores, they are all usually simple and do not have any intelligence in the way they execute instructions. They can also not work on different activities. If one core in the GPU works on a particular matrix multiplication, the other cores cannot work on matrix addition, etc. This is the very essence of the SIMD architecture that the GPUs are all based upon. It is the same uniformity of executing instructions that also makes them fast. FPGAs are actual hardware. The hardware can be programmed in any manner that is desired and multiple processes can reside inside the same FPGA at the same time in real time. This is contrary to how CPUs work. CPUs perform context switches to switch between different processes and time share the processor. CPUs are also much more clever compared to FPGAs. So the introduction of an FPGA essentially bridges the gap between CPUs and GPUs in many ways, thus making a more concrete computation platform.

I feel personally that it is the correct time for this research to go ahead as there is a lot of industry experience in adapting GPUs into CPUs and making a good hybrid processor that has become so common in all laptops and desktops. Even supercomputers are integrated with such nodes and are seeing a massive boost in compute power because now, each processor has its own specialty and is thus highly optimized. In such a time period, integrating FPGAs would be more appropriate as it bridges the gap between the CPUs and GPUs by many ways.

1. It is real time and achieves cycle wise accuracy in terms of time. CPUs with operating systems do not achieve this due to the time sharing of processes and there is no real time guarantee. FPGA being physical hardware can share various processes on various parts of the FPGA at the same time.
2. It is dynamically reconfigurable. Previous generations of FPGAs needed a lot of time to reconfigure and it could not be done on the fly dynamically. Now, FPGAs have the capacity to dynamically change their functionality. So, one can choose to adapt 10 different processes on the FPGA or one extremely heavy process or a combination of those.
3. It can be memory mapped. FPGAs can be memory mapped directly onto the processors address space. GPUs are connected by IO interconnects which are much slower than memory and is more like communication between two processors. FPGAs can be a part of the CPU memory and can freely interact like how shared memory processes can. In fact there is no need for any special control to control an FPGA that is memory mapped.
4. FPGAs can be integrated at the cache level with the processors too. FPGAs are faster than the L2 cache that most processors have. So an single FPGA can change into the processor cache, it can become a memory controller, it can expand the memory of the process when the application needs it. While doing all of this, FPGAs can also run other applications in parallel.
5. FPGAs high very high amounts of pins that come out of the package. It could be used to interface with a lot of IO modules and also many RAMs so that the FPGA can behave as the RAM conrtoller and directly feed memory the processor. The most interesting thing in all this is that the memory controller can be upgraded anytime since the FPGA is programmable. We can have different L2 cache sizes and configuration. These changes can never be performed on a real processor without modifying the core of the processor design and is extremely costly.
6. FPGAs can morph into any external device that a processor wants. If other GPUs are mounted on the board, FPGAs can interface them with the processor, it can connect other boards with the processor. It can directly interact with other GPUs that are present so that the CPU is not interrupted.

Essentially apart from running computations, FPGAs become the tool for modifying a lot of processors' architecture specially in the memory hierarchy side. They are very good at being low power devices due to their adaptation in the embedded market. It can be all of those mentioned above at the same time because different parts of the FPGA hardware different functionality can be implemented.

Despite all these advantages, the industry has not attempted such a design yet. There are reasons for the same. Firstly, CPUs, FPGAs and GPUs are totally different types of designs. Companies that are good at one are not yet good at the others. Without the outrageous success of NVIDIA, currently Intel is working on GPUs at full throttle. It was not a natural transition for them. Similarly, Xilinx, a leading FPGA production company has been trying to integrate CPUs with FPGAs. Since they are not a processor company, they use open source, smaller and less powerful processors and they have been successful in this sense. Time is right now for the computer architecture community to consider the integration of both these models to derive a better and more optimized machine that is not only enormously fast but also runs the right algorithms on the right platforms that they are best suited to.

3. PROJECT PLAN

The goal of this proposal is to analyze deeply if the inclusion of FPGAs on chip can delivered better performance and benefits compared to the existing architectures. To achieve this, we propose developing a library suite, much on the same lines as NVIDIA CUDA to enable quick translations to FPGAs and easier job offloading. We identify the various family of algorithms that run faster on FPGAs and identify the limitations of FPGAs in terms of task characteristics.

Once a basic analysis is done, comparative studies with implementation on GPUs and CPUs will be done. Following this, using benchmarks, the occupancy of these processing devices for various applications will be studied. FPGAs will then identified for a specific list of tasks and the observed benefit will also be analyzed. Many critical technologies like mission computers and real time RADARs, observatories use FPGAs for concurrently managing several tasks in quick time with cycle-wise accuracy that generally processors cannot guarantee with operating systems sitting on top of them. FPGAs are also known for their large pin outs thus serving as good IO controllers and memory controllers. They can also bridge the gap between CPUs and memory but serving as intermediate memory and prefetch mechanisms that some CPUs do not inherently support.

The dynamic reconfigurability is central to such optimizations and mechanisms for clever reconfiguration are also to be implemented so that the FPGA is not always interrupted with reconfiguration commands from the processor.

After undertaking these exercises, issues like resource contention, performance benefits of executing other programs compared to the currently active ones on the various cores and managing job handling and offloading automatically, modifications to the host scheduler have to be tackled in order to be able to perform resource allocation to the individual units like they are elements of resources. In addition to clever algorithm, interfaces between the CPU and FPGA also need to be establish assuming a type of high speed interconnect or by using the traditionally available interconnects on the FPGA and extrapolating such results for on chip interconnects. Other aspects of memory sharing and low power modes which could be effectively implemented on FPGAs will also be looked at.

Based on various papers on feedback control system, heart beat based performance modeling, monitoring and controlling of devices will also be incorporated to make the system whole and worthwhile considering for future usage. Without feedback control mechanisms, the performance and power will both be badly affected and below optimum levels.

4. BENEFITS

Benefits of such an exercise are multifold.

1. We can develop extensive library support for FPGAs, which can be used for more development and libraries that we can use to control FPGAs from CPUs irrespective on their presence on a dye.
2. Increased coverage for algorithms with platforms as diverse as the CPU/GPU/FPGA combined platform. With careful resource allocation, one can also experience better

performance per Watt and less resource contention and dependency on one communication medium.

3. Increased concurrency. The magnitude of the increase will be dependent on the programs and the type of system and application level utilities that are in place.
4. Low power modes are very successful on FPGAs and cycle time accuracy can be maintained with FPGAs.
5. FPGAs can serve as additional level of cache / expandable memory controllers to access FPGAs memory and could be equipped with very good prefetch logic that could be controlled or programmed either as upgrades or by the OS depending on type of applications. Computer architects can also build additional hardware support that would previously involve modifying the processor core.

5. BUDGET / TIME FRAMES

The aim of this project is to study all the above mentioned points and their feasibility in an estimated time period of 4 years. The PI has worked on all the platforms before and has also been a part of a hardware board design that features multiple CPUs and FPGAs on the same board. He has also worked on GPUs hosted by CPUs for a few years and published papers on the same. The project will be completed in the following steps.

1. First year would investigate the memory mapping mechanisms that CPUs and FPGAs interact and the overheads of the mechanisms and analysis of various types of algorithms suited for FPGAs

2. In the second year, research on algorithms suited for the FPGAs, GPUs and CPUs will continue with more detail on the exact performance gains and a criteria will be devised that decides when to allot an algorithm to either the CPU or GPU or to the FPGA.

3. Compiler and dynamic libraries will be developed for switching between various tasks on the FPGA in real time and reconfiguration of the FPGA will be investigated for running multiple processes and also for removing some processes, adding processes and killing processes.

4. The versatality that is added when FPGAs are closely integrated with CPUs will be looked at. The research until now will be performed with memory mapping on the RAM level. i.e. FPGA accesses will be as fast as RAM. FPGAs will be coupled to processor simulators to facilitate a closer integration so that cache configurations, coherency configurations and various other architectural parameters can be employed in the FPGA, making the FPGA an integral part of the CPU processing chain.