Abstract: Architectures and Compilers for Tiger Machines

In this abstract we introduce *Tiger* processors, **TI**ghtly couple a **GE**neral-purpose core with a **R**econfigurable computing device. Tiger offers the best of two worlds: the flexibility of a general-purpose microprocessor and the performance of a custom circuit. By combining our compiler techniques with this new architecture we show that Tiger processors deliver increased performance across a broad range of metrics — throughput, memory utilization, manufacturing cost, design complexity, fault tolerance, and power consumption — on chips of the future.

Of the many goals in modern processor design, first and foremost is that the chip must deliver increased performance despite the increasing gap in speed between the memory system and the processor. The performance must scale with the many hundreds of millions of transistors that will soon be available to chip designers, without increasing design or testing costs. The chip must support forward compatibility, i.e. as new generations of the chip appear, performance of an application must increase without requiring its redesign.

The chip must deliver this performance over a large range of applications particularly because the application base has broadened significantly to include areas such as multimedia, encryption, signal processing, object recognition, and autonomous vehicle control. The range of computing platforms is broad as well. Processors are expected to operate efficiently in workstations, laptops, PDAs, set-top boxes, and even satellites.

These demands on processors have their corresponding requirements in compilers. They must be able to produce efficient executables for a variety of platforms from any high-level programming language. Economic pressures are shortening the design cycle of both desktop and embedded applications, requiring a technology that allows a single source program to become an application on a broad spectrum of computing environments.

Reconfigurable fabrics offer high performance and efficiency for many applications because they can be configured to implement hardware matched to each application as needed. Furthermore, the configurations are created at compile time eliminating the need for complex control circuitry. Because the compiler can examine the entire application, a reconfigurable device can exploit all of an application's parallelism: MIMD, SIMD, instruction-level, pipeline, and bit-level. The number, type, and size (i.e, width) of the function units can be tailored directly to the application. Not all applications, or all parts of a single application, lend themselves to being run on a reconfigurable device, however.

Tiger processors, which combine a reconfigurable device with a general-purpose processor, are an attractive means by which processor architects can increase the performance of microprocessors for a broad range of applications at a limited cost. On such a machine the processor executes the portion of an application that is not suited for a reconfigurable device and acts as a fall-back, so that any application can be executed. The reconfigurable fabric augments the processor with application-specific hardware. Its regular nature decreases chip design complexity and testing costs while increasing yield. Tiger machines can be used in both the desktop and in embedded systems. In fact, by combining dynamic built-in self-test and dynamic n-way fault tolerance Tiger allows one source to create applications with varying levels of fault-tolerance—fault-tolerance which can be tuned at runtime, trading off performance, power, and fault-tolerance.

The two things that enable the compiler to effectively partition applications are the close coupling of the fabric and the core and advanced configuration loading instructions. Close coupling allows small blocks of code within tight dependencies to be placed on the fabric. The compiler uses configuration prefetching to reduce the configuration load times, and speculative configuration load instructions are used to increase the number of potential pieces of an application which can be run on the fabric (also providing forward compatibility). Loops are turned into hardware pipelines using a modification of software pipelining which uses read-read dependencies to ensure the use of short local wires on the fabric. For example, a FIR filter can be partitioned and pipelined, resulting in a substantial performance increase over traditional FPGAs and DSPs (see Fig. 1). PGP gains a conservative 12x improvement over current processors by decreasing memory bandwidth requirements (both compression and IDEA are placed on the fabric), decreasing circuit complexity (appropriately sized function units), increasing parallelism (10s of adders, multipliers, etc.), and propagating constants (decreasing circuit complexity). Both of these applications are stream-based. On the other hand, Mergesort takes advantage of the close coupling to obtain speedup (See Fig. 2) from both decreased memory bandwidth requirements and increased parallelism.

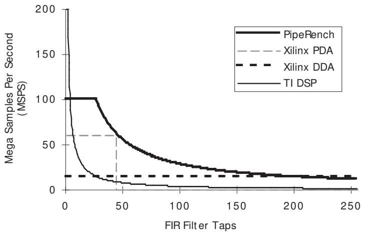

Figure 1. 8-bit FIR filter Performance

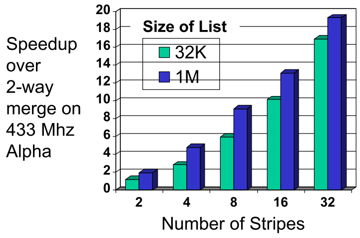

Figure 2. Mergesort Performance