Building a Better Bathtub: Computing at the Optical Memory Interface

Donald Chiarulli[a], Steven Levitan[b], Robert Hofmann[c]

[a]Department of Computer Science, University of Pittsburgh
[b]Department of Electrical Engineering, University of Pittsburgh
[c]Department of Physics, University of Pittsburgh

ABSTRACT

To fully exploit the high bandwidth and inherent parallelism of optical memory systems, it is necessary to perform correspondingly parallel computations at or near the interface to the memory system. In this paper, we present a system in which a dynamically reconfigurable processor is built at the optical memory interface. Dynamically reconfigurable processors exploit parallelism at the level of individual machine instructions. They are based on the time multiplexing of gate array logic between various processor configurations, each of which is matched to a particular required computation. This paper is an analysis of the performance of an optically reconfigurable processor in comparison to conventional multiple instruction issue processors. We will show that the volume of configuration data required makes these systems difficult to build in electronic implementations but ideal for implementations with optical memory.

Keywords: optical memory, optical memory interface, reconfigurable computing, reconfigurable processor

1. INTRODUCTION

Page-oriented volume optical memories[1,2] hold the prospect of high spatial and temporal bandwidth at the optical memory interface. However, limitations in communication bandwidth and parallelism in electronic systems make it difficult to process this data at any significant distance from the optical memory device. Additionally, in order to take advantage of the inherent of parallelism of optical memory access, a corresponding degree of computational parallelism must be available. For this reason, optical memories are typically associated with applications and systems such as image processing, where a high degree of single instruction multiple data stream (SIMD) based parallel processing can be supported. This is analogous to using a fire hose to fill a bathtub.

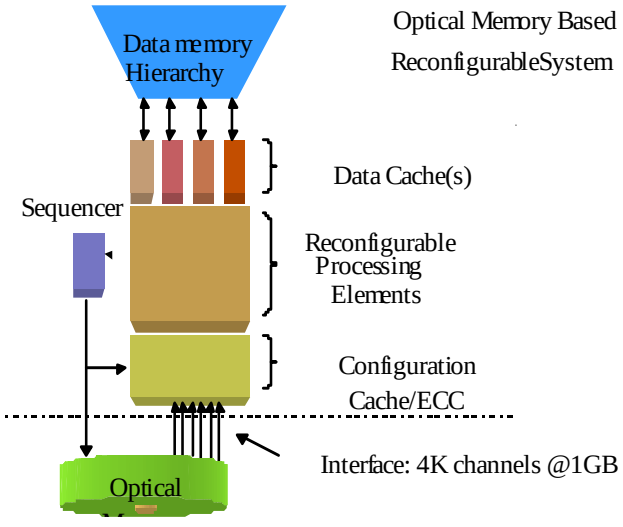

Figure 1: Optical Memory Based Reconfigurable Computing System

In this paper, we will analyze an alternative architecture in which a dynamically reconfigurable processor is incorporated into the optical memory interface. Dynamically reconfigurable processors are designed to exploit instruction level parallelism (ILP). This technology is based on reconfigurable logic techniques such as those that are widely used in field programmable gate arrays (FPGA's)[3]. However, unlike one-time programmable FPGA's, the processing elements are dynamically configured in sequence during a computation. Each configuration attempts to exactly match the operations and interconnection network required for a specific region within the application's control-data flow graph. Thus, deep pipelines and maximal ILP are possible. The bottleneck in these systems is the large volume of configuration data that must be delivered to the chip at high bandwidth. Optical memory is the ideal media to break this bottleneck.

2. INTERFACE ARCHITECTURE

Figure 1 illustrates an optical memory interface based on a reconfigurable computing platform. In this system, computations are performed directly at the optical memory interface. Unlike a conventional processor, in which dedicated control and arithmetic/logical hardware interpret a fixed set of instructions to perform a desired computation, this system implements the same computation as a sequence of configurations to an array of reconfigurable logic blocks. A sequencing unit is responsible for ensuring that configuration data is read from the optical disc into the cache when needed and for presenting configurations to the processing element in the correct order. There is a separate, conventional, memory hierarchy that provides the processing element with access to the target data for the computation.

To demonstrate the viability of this design, the focus of this paper will be a comparative analysis of the performance of a fixed functional unit processor that supports ILP, such as a superscalar or superpipelined machine, versus the performance of corresponding reconfigurable processor. In the case of the fixed functional unit processors, we will make no assumptions about the architecture or program behavior except to assume a specific level of ILP. This can be interpreted as ideal behavior on a minimal set of resources or as effective ILP on a machine with greater capabilities. Our only interest is to make a determination of the characteristics of a reconfigurable processor that can achieve the same thoughput.

3. PERFORMANCE MODELS

1. Fixed Function Processor Model

We begin our analysis with the well-known performance equation for a VonNeuman architecture shown below.

*Run Time = Dynamic Instruction Count * Average Clock Cycles per Instruction*

We are measuring performance as the run time of a specific program, expressed as the number of instructions executed in the course of a computation times the average number of clock cycles required for each instruction. Since in this analysis we will compare this value to reconfigurable processor, we have chosen to omit the technology dependent clock period term and instead express run time in the more generic unit of clock cycles. In this equation, instruction level parallelism gained from superscalar or superpipelined designs is incorporated in the *Average Clocks per Instruction* term. Abbreviated to $CPI_{avg,}$ this term can be rewritten as follows

$$CPI_{avg} = (T_f + T_e)/(D * ILP)$$

where T_f and T_e are the times (in clock cycles) required to fetch and execute the instruction respectively, D is the pipeline depth of the instruction execution pipeline and ILP is the number instruction pipelines operating in parallel on average.

2. Reconfigurable Processor Model

On a reconfigurable processor, instead of a sequence of instructions, the computation consists of a sequence of logic configurations, each of which is applied in turn to the reconfigurable logic array. Thus, we model the run time of a reconfigurable processor as

*Run Time = Dynamic Configuration Count * Average Clock Cycles per Configuration + Configuration Re-use*

The Average Clocks per Configuration, CPC_{avg}, is the average time between configurations in the program sequence. It consists of the time to load the configuration, T_R, plus the time required to clock a computation through the logic in the configuration, T_E. Ignoring T_R for the moment, T_E is dependent on two factors, the size of the configuration, $M_{config,}$ and the number of operations it contains that can be done in parallel, the ILP. If T_m is the number of clock cycles required per operation and M_{config} is the number of operations in a configuration, then CPC_{avg} can be expressed as

$$CPC_{avg} = T_R + T_E = T_R + (T_M * M_{config})/ILP$$

Unlike CPI_{avg} in the previous analysis, the CPC_{avg} term does not capture all of the available parallelism in a reconfigurable system. Consider a case where an entire body of a loop can be incorporated as a deep pipeline in a single configuration. In this case, the configuration itself becomes an instruction level pipeline and can be reused without reloading. Taking into consideration the total number of operations in an application, M_{prog} , the size of a configuration and the average configuration reuse, L, the dynamic configuration count can be expressed as:

$$DCC = (M_{prog} / M_{config}) - L$$

Comparing this model to the model for fixed function processors from the previous section tells us two things about the relative performance of reconfigurable systems. First, both designs attempt to exploit instruction level parallelism with the impact measured in more or less the same way. In other words, two machines with the same ILP would gain approximately the same benefit. To the extent that reconfigurable processors are expected to support greater ILP by more closely matching the processor structure to the computational flow graph, this is a positive result. However, this difference may not be significant and will certainly be application dependent. The greater performance improvement comes from deep pipelines formed by incorporating loops into single configurations, shown as the reuse, L, term above. Each configuration reuse adds only a single clock cycle to the total run time while it reduces the *DCC* times CPC_{avg} product by CPC_{avg} clock cycles.

In either case, this analysis has ignored a relative cost of instruction fetch time, T_I, on fixed processors and reconfiguration time, $T_{R,}$ for reconfigurable processors. Computer architects have long understood how to hide the latency of instruction fetches using cache memories. Although a similar approach can be taken for T_R, the number of bits required to encode a configuration is typically two to three orders of magnitude greater than a typical machine instruction, even considering the multi-instruction fetches used on modern processors. In the next sections we will discuss methods for hiding reconfiguration latency and analyze the performance impact of these techniques.

4. EFFECT OF ON-CHIP CONFIGURATION CACHE

The initial approach to hiding the configuration latency of reconfigurable processors is basically the same as with a fixed processor, the addition of an on-chip configuration cache. This cache is distributed among the logic blocks and allows the processor to switch between the current configuration and a cached configuration in a single clock cycle. However, if the new configuration is not in cache, then the processor must load it from off chip at a far greater latency cost. Thus, if P_{hit} is the probability that the next configuration will be found in cache, the average reconfiguration time can be written as the weighted average of the on chip configuration time, one clock cycle, and the off chip configuration time as follows

$$T_R = P_{hit} + (1 - P_{hit})(B_{config} / N)$$

In this expression the off chip configuration latency is expressed as the size of the configuration in bits, B_{config}, divided by the number of available configuration data channels, *N*.

Cache size for a reconfigurable processor is a significant issue. On one hand, increasing the cache size will improve performance by increasing P_{hit}. On the other hand, any increase in the cache size reduces the available number of logic blocks. The reduction of processing area will result in smaller configurations, which may decrease performance by increasing the dynamic configuration count, by reducing the amount of instruction level parallelism, and limiting the size of loops that can be captured into reused single configuration pipelines. It is necessary therefore to perform a careful analysis comparing the gain in performance due to the increased hit rate to the loss in performance due to the reduced processing area in order to determine the optimal cache size.

5. PERFORMANCE COMPARISONS

1. Performance estimates based on electronic reconfiguration

Given these characteristics, in this section we will analyze the performance of a reconfigurable processor with various amounts of on-chip configuration cache. Since we will eventually compare this data to fixed functional unit machines, we will make an assumption that each operation executed on the reconfigurable machine directly corresponds to one instruction on a conventional machine. This allows us to devise a comparable metric to CPI for reconfigurable machines which we call effective or CPI_{Eff} . Effective CPI is the run time of a program on a reconfigurable machine divided by the number of instructions in the original program prior to conversion to logic configurations.

$$CPI_{Eff} = \frac{Run\ Time}{\#\ of\ Instructions}$$

An inaccuracy introducted by our assumptions about the mapping of instructions to operations will be in favor of the fixed functional unit machines since we typically would not directly map transfer of control instructions.

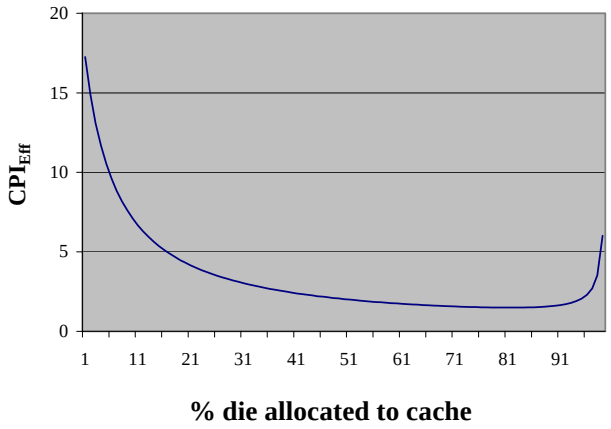

Figure 2: Effective CPI vs % cache for a reconfigurable processor with N=128 channels

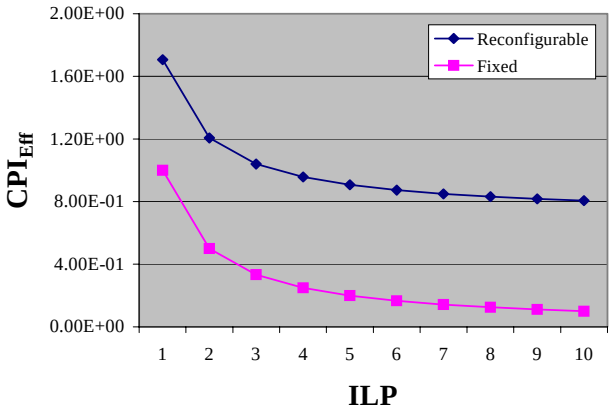

Figure 3: Effective CPI vs ILP for a reconfigurable processor with N=128 channels

In Figure 2, we have plotted the CPI_{eff} vs. cache size for a reconfigurable processor implemented on a chip with a capacity of 10^4 gate equivalents and 128 configuration channels. The program is assumed to have an average operation size of M_{op} = 500 gate equivalents, and an *ILP* of 1. The cache size was varied from 0% to 99% and the hit rate for this analysis was computed based on a model of locality known as the Parachor Curve[4]. The Parachor Curve is computed based on a locality metric for the program, .95 in this case, and the ratio of processing area to cache area on the die.

In Figure 3, we have taken the electronically cached reconfigurable processor at 76% cache and plotted its performance vs. a conventional superscalar processor as the ILP is varied from 1 to 10. This graph is significant, it tells us that under these constraints, even significantly greater ILP in the reconfigurable case cannot generally outperform a conventional fixed functional unit processor. At 128 configuration channels, the miss penalty for reconfiguration requires that optimal performance can only be obtained when most of the resources on the chip are used for cache. With so few resources given over to processing, configurations become so small that *ILP* contributions to performance enhancement are small. It is clear that in order to make reconfiguration profitable it must be made faster and that the addition of configuration cache alone is insufficient to meet our goals.

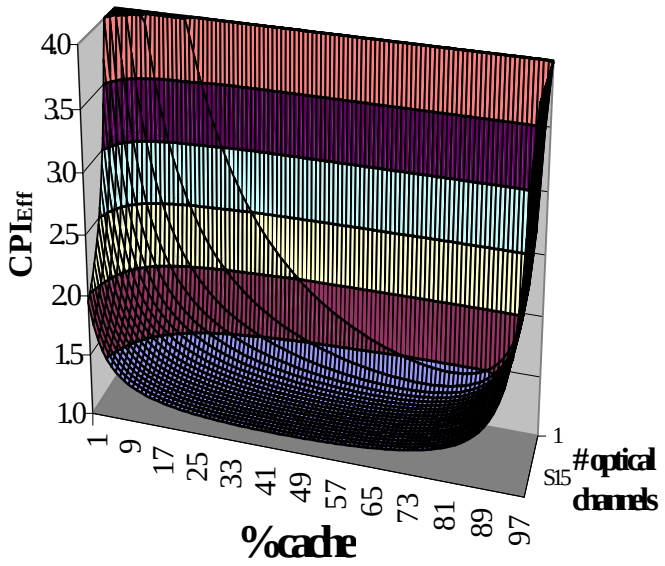

Figure 4: Effective CPI vs. % cache vs. # of optical channels for an optically reconfigurable processor

2. Performance estimates based on optical reconfiguration

From the previous analysis, it is clear that for a dynamically reconfigurable processor to be viable, we must further reduce the reconfiguration time by increasing the number of channels available for configuration data. The introduction of optical reconfiguration from a page-oriented optical memory makes this possible. Figure 4 is a 3-D plot of the Effective CPI of a reconfigurable processor as the %cache is varied from 0 to 100% and the number of configuration channels is varied. from 1 to 4000. From this graph we have extracted a single plane at 40% cache and plotted it in Figure 5. From this slice, we can see that given sufficent reconfiguration channels the *Effective CPI* can be made to approach one for a benchmark with an ILP of one. In this case, good results can be obtained with a optical page size of 64x64 bits, or about 4000 optical channels.

Finally in Figure 6, , we have replotted the *Effective CPI vs. ILP* curve at 40% cache and 4000 optical

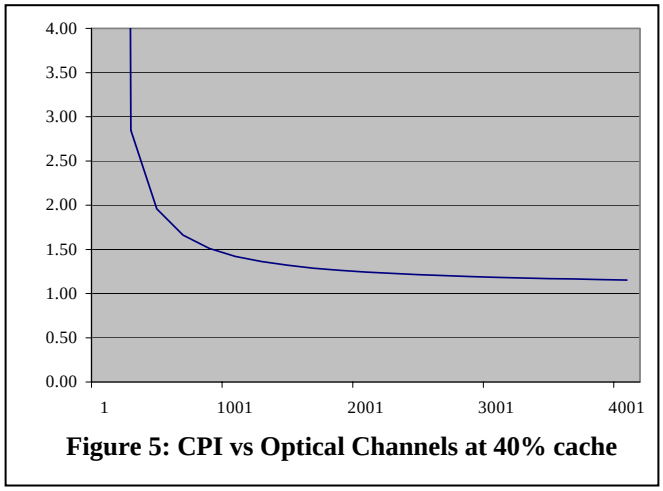

Figure 5: CPI vs Optical Channels at 40% cache

channels and compared it with the conventional processor. Under these conditions, the two systems track closely as ILP is increased. Thus small improvements in ILP based on dynamic reconfiguration can lead to better performance. However, the best is yet to come, we will now consider the additional performance increases from pipelined configuration reuse.

3. Effects of Pipelined Reuse

In Figure 7, we have taken the optimized optically reconfigured processor from the previous analysis, with 4000 optical reconfiguration channels and 40% of the die area reserved for configuration cache and examined its performance for configuration reuse values at 0%, 50%, 75% and 90% of all executed configurations. It is seen that even at a moderate reuse estimate of 50%, the reconfigurable processor outperforms the superscalar processor at *ILP*'s less than 5. If 90% reuse is achieved, then the reconfigurable processor display better performance over the entire range of ILP. The most dramatic improvement over 0% reuse estimates occurs at low levels of *ILP*. At low *ILP*, the depth of the average configuration is longer, leading to a larger gain in performance from pipelined execution.

6. CONCLUSIONS

As we have shown, optically reconfigurable computing platforms can offer the potential for increased performance over even an optimistic estimate of superscalar processing performance. Speed increases due to the exploitation of ILP alone can nearly match those that are seen in the behavior of superscalar architectures. Reconfigurable processors clearly gain the advantage, however, when they capture an entire loop in a single configuration and reuse it in an instruction level pipeline.

In order to capture large loops and maximize the use of available ILP, reconfigurable processors with very large amounts of available processing logic and low reconfiguration latency are required. Increasing the amount of processing logic is a two-edged sword. While it increases performance by better capturing available *ILP* and *Pipelined Reuse*, the corresponding increase in configuration data requirements easily overwhelm any gains.

We have shown that the number of configuration data channels necessary to provide a satisfactory temporal and spatial bandwidth to the reconfigurable processor is at least an order of magnitude greater than is currently available through purely electronic means, even with aggressive use of on-chip cache memory. Integration of the reconfigurable processing elements directly into the interface of a page-oriented optical memory offers a perfect solution to this problem. At four thousand channels, equivalent to a 64 bit x 64 bit page of optical data, we have shown that a reasonably sized reconfigurable processor with modest estimates of *Pipelined Reuse* offers a significant increase in performance over a modern superscalar processor.

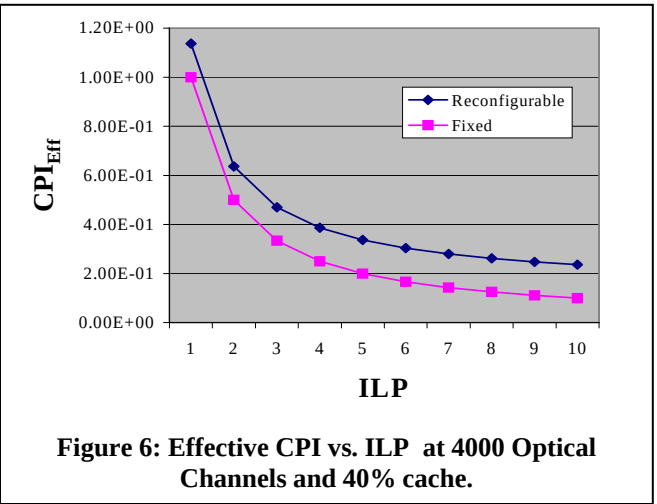

Figure 6: Effective CPI vs. ILP at 4000 Optical Channels and 40% cache.

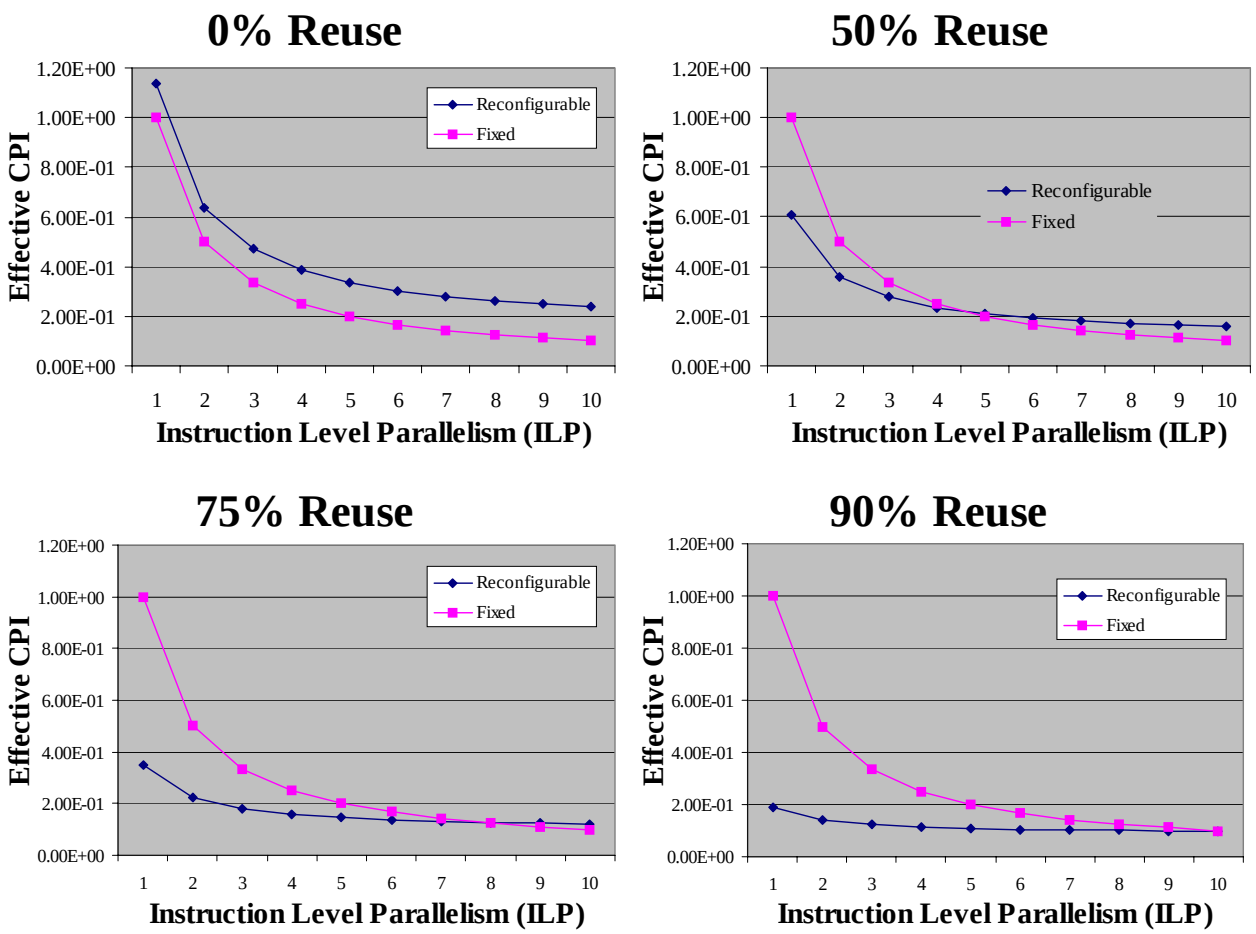

Figure 7: Comparison of Effective CPI for reconfigurable processor vs. conventional processor for varied ILP and Configuration Reuse values of 0%, 50%, 75%, and 90%