

Empirical Analysis of Operand Usage and Transport in Multimedia Applications

Hongkyu Kim, D. Scott Wills, and Linda M. Wills
School of Electrical and Computer Engineering
Georgia Institute of Technology
{hongkim,scott.wills,linda.wills}@ece.gatech.edu

Abstract

As transistor feature sizes decrease exponentially, the critical problem in massively parallel architectures for multimedia systems becomes the wire delay latency imposed by transporting operands. In this paper, we first characterize the locality properties of operands in multimedia applications to motivate the development of alternate low-cost and low-latency operand communication mechanisms. Based on the locality properties, we then present the results of a simulation study that estimates the impact of varying sized local storage on the transport complexity of the operands. A small amount of local storage (eight-entry) per functional unit with approximate information on operand dynamic lifetimes is sufficient to suppress 79.4% of operand writes in MediaBench applications. The empirical study shows that local storage alone reduces operand read traffic by upto 35.5%.

1. Introduction

As the feature size in CMOS technology approaches 100nm, the promise of giga-transistor chips with multi-gigahertz clock rates moves ever closer. The trends from the International Technology Roadmap for Semiconductors (ITRS) suggest the need to focus less on “transistor-centric” design and more on “interconnect-centric” techniques since interconnect delay does not scale as well as gate delay [1]. At the architectural level, this focus shift translates to less attention on functional unit-centric design and more attention on operand transport. For instance, in conventional superscalar microarchitectures, increasing the resources to improve the instruction per cycle (IPC) throughput is limited by the latency to transport all necessary operands to the functional units (FUs) that need them. The architectures suffer from long interconnects required to carry operands between a storage unit – a register file or a memory bank – and FUs; or through an operand bypass network.

From the architectural viewpoint, the straightforward way to solve the interconnect bottleneck is to shorten the distance of operand transport i.e., reduce usage of the

global interconnect and convert the global communication into point-to-point local communication. Toward this goal, we analyze application programs to understand their data communication needs. This involves modeling operand usage and the operand bandwidth of individual components. The purpose of the analysis is to get a better understanding of the usage and the transport properties of the operands. The operand characteristics extracted from the analysis can be exploited in the design of mechanisms to optimize operand delivery.

The rest of this paper is organized as follows. Section 2 overviews prior work. Section 3 introduces the empirical study by defining terms and metrics used and by presenting our methodology. Data on the properties of operand’s localities and the result of an empirical study of the operand’s traffic appear in Section 4. Finally, Section 5 summarizes conclusions.

2. Related Work

Many prior approaches have attempted to optimize the operand transport for wide issue processors. Early work by Franklin and Sohi examined the temporal locality properties of register references to reduce centralized register file traffic [2]. But they confined their study only to the register references. We extend this work by examining a broader set of operands and study the spatial as well as temporal locality of operands. Spatial locality is as important as temporal locality since the movement of operands is determined by the physical location where it is produced and consumed.

Recently, new architectures have been proposed to reduce the complexity of operand transport. Burger and Keckler presented Grid Processor Architectures (GPAs) that map compiler-scheduled blocks onto a two-dimensional grid of ALUs [3]. The strategy is to localize the operand communication by forwarding temporary values generated inside a basic block directly from the producers to their consumers. Corporaal proposed a new architecture based on the transport-triggering concept which is programmed by explicitly specifying the data transports [4]. It directly forwards operands between FUs and reduces the latency of bypass network by eliminating

the associative hardware. All bypassing is done in software under compiler control.

An alternate approach is to change the organization of FU and the hierarchy of register file to reduce the operand traffic dynamically. A fully distributed register file [5] with a dedicated register transfer network can be employed to reduce operand access time and to eliminate the need for broadcasting operands. It can only access the local registers assigned to the cluster. An explicit register transfer operation delivers the operand through a partially connected network if an inter-cluster transfer is needed. This work demonstrates the effectiveness of local storage in reducing operand transport. This paper presents an assessment of the actual communication resources needed.

3. Methodology

In this section, we explore the characteristics of operand usage and transport properties in multimedia applications. The MediaBench programs were compiled using the PISA (Portable Instruction Set Architecture) gcc cross compiler in the *SimpleScalar* toolset [6]. The *sim-safe* simulator was extended to handle operand-based operations instead of traditional register-based operations. All data are measured from the trace driven simulation of dynamic instruction stream. An operand is defined as a value in a register, a memory location, or a memory address. Two kinds of operand characteristics were analyzed: (i) which instructions consume operands after they are produced (temporal locality), and (ii) from/to which FU operands are moved in the execution model (spatial locality).

Temporal locality: We adopt temporal locality metrics defined by Francklin and Sohi [2], but we apply them to memory as well as register operands. We examine *degree of operand use* (the number of times an operand is consumed), *operand lifetime* (the distance in instruction between an operand creation and its last consumption), and *operand age* (the distance in instructions between an operand creation and its first consumption).

Spatial locality: An operand should be transported from its producer to its consumers. The *degree of functionality* is defined by the number of FUs that use the operand. The higher degree of functionality an operand has, the more it will be transported. Actually, this metric depends on the configuration of FUs including the functionality of each FU and the number of FUs. We assumed five heterogeneous FUs – load/store, branch, integer ALU, integer multiplier, and floating-point unit.

Operand Transport: The *operand read transport rate* ($trans_{rd}$), and *operand write transport rate* ($trans_{wr}$) are defined as the following.

$$trans_{rd} = \frac{\sum_{trace} Transported\ Read}{\sum_{trace} Read}, \quad trans_{wr} = \frac{\sum_{trace} Transported\ Write}{\sum_{trace} Write}$$

These metrics depend not only on the configuration of the FU, but on the configuration of storage. In this study, two levels of operand storage hierarchy are assumed: global buffers and local buffers attached to each FU. A read transport occurs when the required operands are found from the local buffer of other FUs or from the global storage. To differentiate two kinds of read transport, we defined one more metric: *bypassed read transport rate* ($trans_{bypass}$). It is defined as follows:

$$trans_{bypass} = \frac{\sum_{trace} Transported\ Read\ through\ Bypass\ Network}{\sum_{trace} Transported\ Read}$$

4. Empirical Result on Operand Usage and Transportation

4.1. Analysis of operand locality characteristics

Figure 1 (a) to (c) show data on the observed temporal locality characteristics for the benchmark programs studied thus far. (*MediaBench* represents average over the benchmark suites studied.) Each bar represents the percentage distribution of each metric of operands. The degree of use is measured in terms of the operand instances, and the age and the lifetime are measured in terms of dynamic instructions. The results show that (i) operands tend to be used only a small number of times – 94.7% of operands are used at most three times, (ii) most operands are first consumed just after they are produced – 82.5% of operands are consumed within five dynamic instructions, and (iii) most operands have short lifetimes – 75.8% of operands are dead within ten dynamic instruction window.

Data was also collected on the degree of functionality of each operand in terms of the number of FUs (Figure 1 (d)). An operand can be used in the same type FU that produced it or in a different type of FU. We differentiated them because if it is used by the same type, it might not be transported and consumed in the producer, but if consumed by the different type, it always moves from the producer to the consumer. Looking at the data presented in Figure 1 (d), a large number of operands in all benchmark are mono-FU operands – an average of 52.3% of operands are used only by the same type FU and an average of 30.5% are used only by a different type FU.

We can infer from the previous observations that most operands exhibit high degrees of temporal and spatial locality. Thus, we can potentially reduce the total distance of operand transportation if (i) instruction results are held in local storage, (ii) they are directly forwarded to their consumers, and (iii) the instruction that consumes the operands can be allocated to the FU nearest the producer.

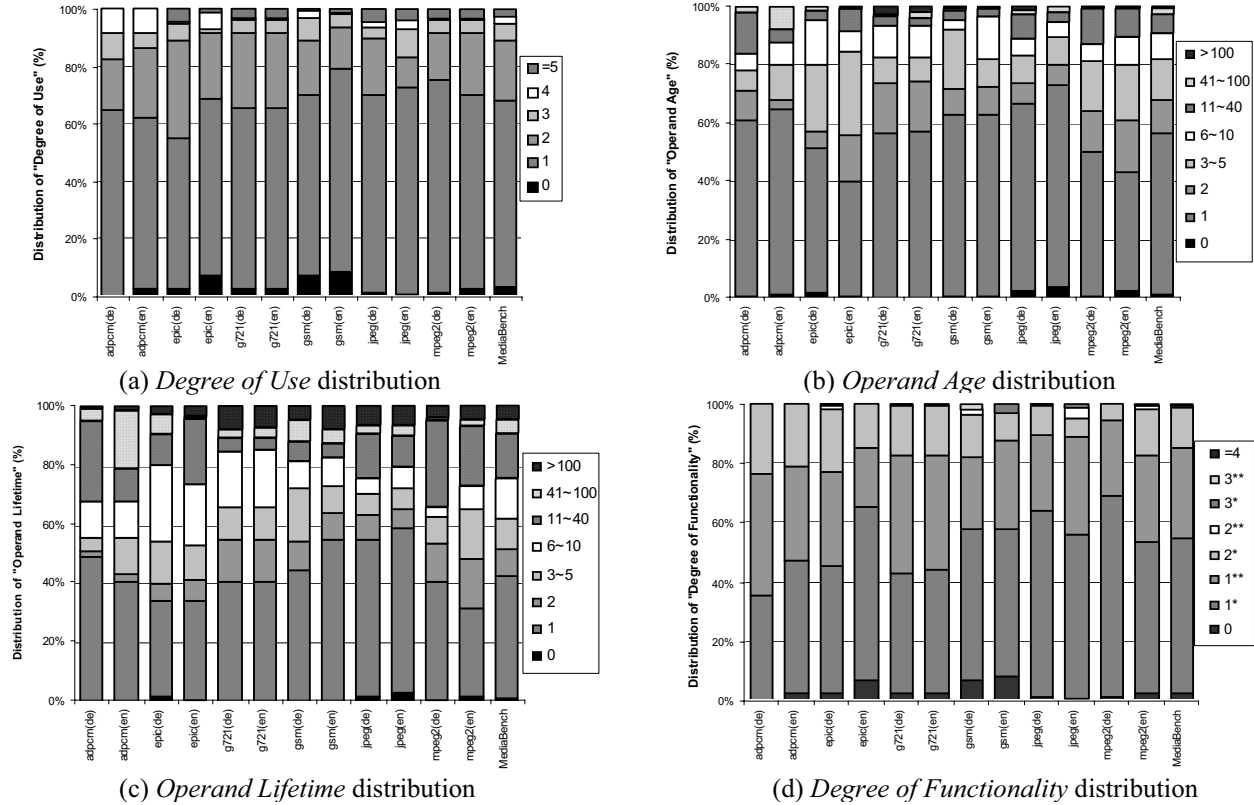


Figure 1. Observed operand locality characteristics. (a), (b), and (c) for temporal locality; (d) for spatial locality. * denotes the number of consumer FUs including producer itself, and ** denotes the number of consumer FUs excluding the producer.

4.2. Evaluating impact of techniques on operand transportation

The temporal locality properties of the operands imply that the local storages in each FU, buffering the results of the last instructions, can reduce the operand traffic. The operand read can be reduced by the short age property and the operand write can be suppressed by the short lifetime property. We hope to determine from our empirical analysis how much local storage and what kind of additional information is needed.

Table 1. Execution model details.

| Pipeline stage | Description |
|----------------|--|
| Read Operand | <pre> if (the operand is in the consumer's local buffer) { else if (the operand is in the other local buffer) { read the other local buffer; copy the operand to the consumer's local buffer;} else { read the global buffer; copy the operand to the consumer's local buffer;} read the consumer's local buffer; </pre> |
| Execution | FU Configuration: 1 load/store unit, 1 branch unit, 4 integer ALUs, 2 integer multipliers, 2 floating point units |
| Write back | <pre> if (local buffer is not full) { write the result in the producer's local buffer;} else { write back the oldest entry to the global buffer; // replacement algorithm = LRU write the result in the producer's local buffer;} </pre> |

To measure the impact of each technique, instruction execution models are devised as shown in Table 1. Figure 2 depicts the model equipped with local storage and fully-connected operand bypass network. FU can access the required operand from (to) its own local storage, one of the other local storage buffers through the bypass network, or the global storage.

Figure 3 presents the operand read and write transport rates for the execution model of Figure 2. The X-axis denotes the number of local buffer entries and Y-axis represents the transport rate. From Figure 3 (a), we can see that the read transport rate decreases as the size of the local storage increases because required operands are more likely to be found in local storage. Collected data indicates that the read transport rate saturates at 35.5% reduction using local storage alone. Figure 3 (c) shows that with a small amount of local storage, a significant number of operands are read from the global storage because of the limited size of the local storage. But as the size of local storage increases, almost all operand are accessed through the bypass network. The read transport rates are dominated by the bypassed transport as the size of the local storage increases.

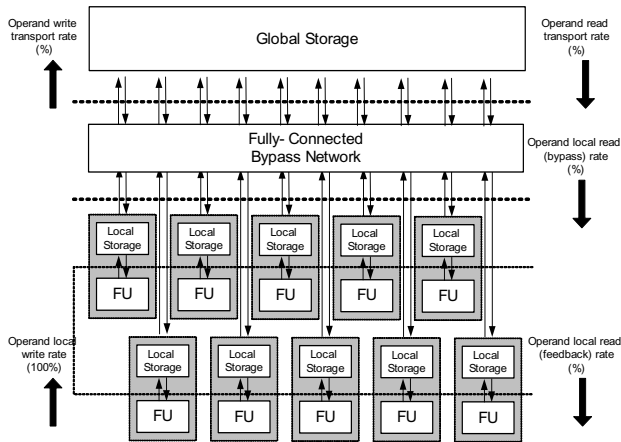


Figure 2. Execution model block diagram.

The write transport rates remain high until a relatively large amount of local storage is available as shown in Figure 3 (b). But most of operand writes are useless because they are unnecessarily written after the operand's lifetime has expired. If the lifetime of each operand is known, the write transport rate can be reduced drastically even with a modest amount of local storage. For example, with an eight-entry local buffer, 79.4% of operands write transport can be removed by using run-time lifetime detection which is based on monitoring register overwrites. If compile-time lifetime detection is used, 95.1% of write transport can be suppressed. This result completely corresponds to the short lifetime property shown in section 4.1. The lifetime detection also slightly reduces operand read because local buffer entries are released after its lifetime. The data in Figure 3 suggest that the lifetime is critical to the write transport rate.

5. Conclusion and Summary

Operand transport demand becomes a limiting factor in improving the performance of modern microprocessors because wire delays make architectures communication

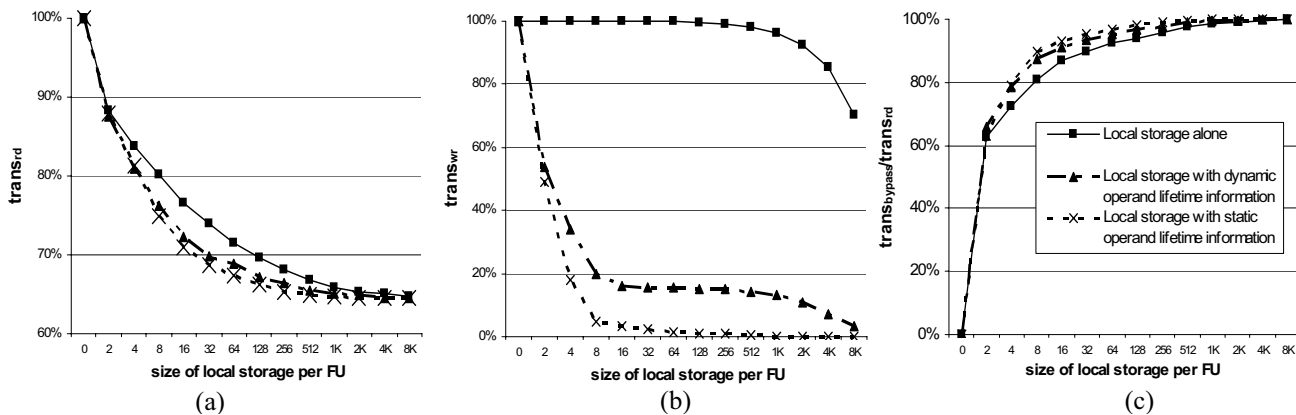


Figure 3. Impact of local storage, fully-connected bypass network, and lifetime detection on operand transport rate. (a) Read transport rate, (b) Write transport rate, and (c) Rate of bypassed read transport over total read transport

bound and prevent scaling of components. To minimize the latency caused by the wire delay, the operand transport rate should be kept as low as possible and the distance that the operand moves should be as short as possible. This paper analyzed the operand usage and transport characteristics in the MediaBench programs. Most operands exhibit high degrees of temporal and spatial locality. These locality properties suggest the use of local buffers to store the operands. We found that the use of the local storage alone can only reduce operand read transport rate. But the local storage with help of operand lifetime information can reduce the operand's write transport rate. With an eight-entry local buffer, 79.4% of operand's write can be removed and 25.1% reads are accessed through the shortest path – the consumer's local storage.

References

- [1] Interconnect Focus Center, "International Technology Roadmap for Semiconductor: Interconnect 2001 Edition." [Online document], Available HTTP: <http://public.itrs.net/Files/2001ITRS/Interconnect.pdf>.
- [2] M. Franklin, and G. Sohi, "Register traffic analysis for streaming inter-operation communication in fine-grain parallel processors," *Proc. Of the 25th Int. Symp. on Microarchitecture*, December 1992, pp.236-245.
- [3] R. Nagarajan, *et al*, "A design space evaluation of Grid Processor Architectures," *Proc. of the 34th Annual Int. Symp. on Microarchitecture*, December 2001, pp. 40-51.
- [4] H. Corporaal, "TTAs: Missing the ILP complexity wall," *Journal of System Architectures*, vol. 45, no. 12, pp. 949-973, 1999.
- [5] S. Bunchua, S. Wills, and L. Wills, "Reducing operand transport complexity of superscalar processors using distributed register files," *21st International Conference on Computer Design*, October 2003, pp. 532-535.
- [6] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, pp. 59-67, 2002.