Instruction Level Distributed Processing: Adapting to Shifting Technology

J. E. Smith

Dept. of Elect. and Comp. Engr.
1415 Johnson Drive
Univ. of Wisconsin
Madison, WI 53706

Abstract. Within two or three technology generations, processor architects will face a number of major challenges. Wire delays will become critical, and power considerations will temper the availability of billions of transistors. Many important applications will be object-oriented, multithreaded, and will consist of many separately compiled and dynamically linked parts. To accommodate these shifts in both technology and applications, microarchitectures will process instruction streams in a distributed fashion -- instruction level distributed processing (ILDP). ILDP will be implemented in a variety of ways, including both homogeneous and heterogeneous elements. To help find run-time parallelism, orchestrate distributed hardware resources, and implement power conservation strategies, an additional layer of abstraction -- the virtual machine layer -- will likely become an essential ingredient. Finally, new instruction sets may be necessary to better focus on instruction level communication and dependence, rather than computation and independence as is commonly done today.

1. Introduction

Processor performance has been increasing at an exponential rate for decades, and most computer users have come to take it for granted. In fact, this performance increase has come only through considerable concerted effort involving the interplay of microarchitecture, underlying hardware technology, and software (compilers, languages, applications). Because of important shifts in underlying technology and software, future microarchitectures are likely to be very different from the complex heavyweight superscalar processors of today.

For nearly twenty years, microarchitecture research has emphasized instruction level parallelism (ILP) -- improving performance by increasing the number of *instructions per cycle*. In striving for higher ILP, microarchitectures have evolved from pipelining to superscalar processing, with researchers pushing toward increasingly parallel processors. Emphasis has been on wider instruction fetch, higher instruction issue rates, larger instruction windows, and increasing use of prediction and speculation. This trend has largely been based on *exploiting* advances in technology and has led to very complex, hardware-intensive processors.

Regarding applications, the focus for microarchitecture *research* has been SPEC-like programs, consisting of single threaded programs written in the conventional C and FORTRAN languages, following a *big static compile* model. That is, an entire program binary is compiled at once, with very high optimization, sometimes

with the assistance of execution profile feedback.

Starting with ever-increasing transistor budgets and the conventional, big-compile view of software, microarchitecture researchers made significant progress through the mid-90s. More recently, however, the problem has seemingly been reduced to one of finding ways of consuming transistors in some fashion. It is not surprising that the result is hardware-intensive and complex. Furthermore, the complexity is not just in critical path lengths and transistor counts; there is also high intellectual complexity resulting from increasingly intricate schemes for squeezing performance out of second and third order effects.

Substantial shifts in both hardware and software technology are currently underway, and the conventional ILP-based microarchitecture approach will not fit with either future hardware or software. In hardware, long wire delays will dominate gate delays, and power consumption is rapidly becoming a limiting consraint. In software, the shift is toward object oriented programs that exploit thread level parallelism and dynamic linking. These shifts will lead to general purpose microarchitectures composed of small, simple, interconnected processing elements, running at a very high clock frequencies. Multiple threads, some completely transparent to conventional software, will be managed by a hidden layer of implementation-specific software, co-designed with the hardware. This hidden software layer will manage the distributed hardware resources to use power efficiently and dynamically optimize executing threads based on observed inter-instruction dependence and communication.

In short, the microarchitecture focus will shift from Instruction Level Parallelism to Instruction Level Distributed Processing (ILDP) where emphasis will be on inter-instruction communication with dynamic optimization and a tight interaction between hardware and low-level software. The following sections are a more in-depth discussion of these ideas.

2. Technology Shifts

First, consider some of the important hardware and software changes that are driving the development of future microarchitectures.

On-Chip Wires

Both short and long wires cause problems for microprocessor designers. In the case of local (short) wires, the problem is one of congestion. That is, for many complex, dense structures, transistor sizes do not determine area requirements, wiring does. With global (long) wires, delays will not scale as well as transistor delays, because wire aspect ratios will be more constrained than in the past and fringing capacitance becomes an important factor.

An interesting analysis of wire delay implications is given in [1] where reachable chip area, measured in terms of SRAM memory cells, is projected. Because of longer wire delays in the future, fewer bits of memory will be reachable in a single clock cycle than today. For example, in 35 nm technology, it is projected that the number or reachable bits will be about half of what is reachable today.

Of course, reachability is a problem with general logic as well. As logical structures become more complex (and take relatively larger area) global delays will increase simply because structures are farther apart. To put it another way, using

simple logic will likely improve performance directly by reducing critical paths, but also indirectly by reducing area (and overall wire lengths). Of course, this is not a new observation -- all the Seymour Cray designs benefited from this principle.

Finally, global multi-stop buses, i.e. buses with several receivers (and possibly multiple drivers) will have very long delays because of both wire capacitance and the loading on the bus. This type of on-chip, intra-processor bus is likely to disappear in the future because of its very poor delay characteristics.

Power

With respect to power, dynamic power is related to voltage levels and transistor switching activity, i.e. dynamic power $\simeq AV^2f$, where A is a measure of switching activity, V is the supply voltage and f is the clock frequency. Higher clock frequencies and transistor counts have caused dynamic power to become a very important design consideration. Because of the dependence on voltage level, the trend is toward lower power supply voltage levels.

The power problem is likely to get much worse, however. To maintain high switching speeds with reduced supply voltages, transistor threshold voltages are also becoming lower. This causes transistors to become increasingly "leaky"; i.e. current will pass from source to drain even when the transistor is not switching. In the future, the resulting static power consumption will likely become dominant, probably within the next two or three chip technology generations.

There are relatively few solutions to the static power problem. One can selectively gate-off the power supply to unused parts of the processor, but this will be a higher-overhead process than clock-gating used for managing dynamic power and can lead to difficult-to-manage transient currents. Alternatively, one can use fewer transistors, or at least fewer fast (i.e. leaky) transistors.

Software

In software, the emphasis in general purpose computing has shifted to integer-oriented commercial applications, where irregular data is common and data movement is often more important than computation. Highly structured data remains important for multimedia applications that tend to use integers and low precision floating point data. These applications are often library-oriented, however, and are often supported by special processors or instruction set extensions.

There has also been a shift toward object oriented languages and dynamic linking which increase programmer productivity and software reliability. Microarchitecture researchers typically assume a high level of compiler optimization, often using profile-driven feedback, but in fact, many application binaries are not highly optimized. Furthermore, global compile-time optimization is not compatible with dynamic linking. And, with a variety of hardware platforms all supporting the same instruction set, it becomes difficult to optimize a single binary to be executed on all of them.

Consequently, the challenge presented to microarchitects is to provide high performance for irregular, difficult-to-predict applications, many of which have not been compiler-optimized.

On-Chip Multithreading

Finally, an important trend that brings together microarchitecture and applications is the use of on-chip multithreading. Multithreading has a very long tradition, but primarily in very high end systems where there has not been a broad software base. For example, multiprocessing became an integral component of large IBM mainframes and Cray supercomputers in the early 1980s. However, the *widespread* use of multithreading has been a chicken-and-egg problem that now appears to be near a solution.

As a way of continuing the exponential CMOS performance curve, single chips now support multiple threads, or will soon [2,3]. This support is either in the form of multiple processors [4] or wide superscalar hardware capable of supporting simultatneous multithreadng (SMT) [5,6]. In any case, the trend is toward widespread availability of hardware-supported on-chip multithreading, and general purpose software will no doubt take advantage of its availability. Furthermore, additional motivation for on-chip multithreading is the increasing number of important applications characterized by many parallel, independent transactions. For example, many web server applications are becoming throughput oriented.

3. Instruction Level Distributed Processing

Historically, computer architecture innovation has not been purely based on exploiting technology advances; it has also been used to accommodate technology shifts. In the case of cache memories, for example, architecture innovation was used to avoid tremendous slow downs - by adapting to the widening gap in relative performance between processor and DRAM technologies. We now seem to be at a point where microarchitecture innovation will be driven by shifts in both technology and applications. The goal is to maintain long term performance trends in the face of increasing on-chip wire delays, power consumption, and irregular throughput-oriented applications that are not compatible with big, static, highly optimized compilations.

A microarchitecture paradigm which deals effectively with technology and application trends is Instruction Level Distributed Processing (ILDP). A processor following the ILDP paradigm consist of a number of distributed functional units, each fairly simple with a very high frequency clock cycle (for example, Fig. 1).

The presence of relatively long wire delays implies processor microarchitectures that explicitly account for inter-instruction and intra-instruction communication. As much as possible, communication should be localized to small units, while the overall structure should be organized for communication. Communication among units will be point-to-point (no busses), with delays measured in clock cycles. Partitioning the processor to accommodate these delays will be a significant part of the microarchitecture design effort. There may be relatively little low-level speculation (to keep the transistor counts low and the clock frequency high); determinism is inherently simpler than prediction and recovery.

With high intra-processor communication delays, the number of instructions executed per cycle may level off or decrease when compared with today, but overall performance can be increased by running the smaller distributed processing elements at a much higher clock rate. The structure of the system and clock speeds have implications for global clock distribution. There will likely be multiple clock domains, possibly asynchronous from one another.

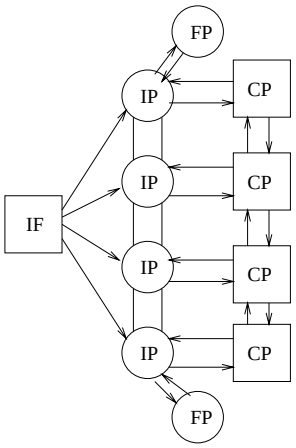

Fig. 1. An example ILDP microarchitecture consisting of an instruction fetch unit, integer processors, floating point processors and cache processors.

Currently, there is an increasing awareness that clock speed holds the key to increased performance. For several years, the big push has been for ILP, and gains have been made, but they now appear to be diminishing, and it makes sense to push more in the direction of higher clock speeds. This idea is not new; the role of clock speed has long been the subject of debate among RISC proponents. Using a very fast clock was certainly the Cray approach, and it is apparent in the evolution of in Intel processors (see Fig. 2). [X11,X12,X13] In contrast to the Intel processors where the pipelines have become extremely deep, the challenge will be to use simplicity to keep pipelines shallow and efficient, even with a very fast clock.

P5 5 stages

Pref	Dec 1	Dec 2	Exec	Wrt Bck

P6 12 stages

F1	F2	D1	D2	D3	Rn	ROB	Rdy/Sch	Disp	Ex	Ret1	Ret2

Willamette 20 stages

IP1	IP2	TC1	TC2	Dr	Al	Rn	Q	S1	S2	S3	Dp1	Dp2	R1	R2	Ex	Flgs	Br	Dr

Fig. 2. Evolution of Intel processor pipelines.

Turning to power considerations, the use of a very fast clock in an ILDP computer will by itself tend to increase dynamic power consumption. However, the very modular, distributed nature of the processor will permit better power management. With most units being replicated, resource usage and dynamic power consumption can be managed via clock gating. In particular, usage of computation resources can be monitored and subsets of replicated units can be used (or not) depending on computation requirements and priorities.

For static power, a high frequency clock will use fast leaky transistors more effectively. If transistors consume power even when they are idle, it is probably better to keep them busy with active work -- which a fast clock will do. In addition, the replicated distributed units will make selective power gating easier to implement. Furthermore, some units may be just as effective if slower transistors are used, especially if multiple parallel copies of the unit are available to provide throughput.

For supporting on-chip multithreading, an ILDP provides the interesting possibility of a hybrid between chip multiprocessors and SMT. In particular, the computation elements can be partitioned among threads. That is, with simple replicated units, different subsets of units can be assigned to individual threads. As a whole, the processor is shared as in SMT, but any individual unit services only one thread at a time, as in a multiprocessor. The challenge will be the management of threads and resources in such a fine-grain distributed system.

Following sections delve deeper into types of distributed microarchitectures.

3.1. Dependence-based Microarchitecture

Clustered *dependence-based* architectures [7] are one important class of ILDP processors. The 21264 [8] is a fairly recent example. In these microarchitectures, processing units are organized into clusters and dependent instructions are *steered* to the same cluster for processing.

The 21264 microarchitecture there are two clusters, with different instructions routed to each at issue time (Fig. 3). Results produced in one cluster require an additional clock cycle to be routed to the other. In the 21264, data dependences tend to steer communicating instructions to the same cluster. Although there is additional inter-cluster delay, a faster clock cycle compensates for the delay and leads to higher overall performance.

In general, a dependence based design may be divided into several clusters; cache processing can be separated from instruction processing; integer processing can be separated from floating point, etc. In a dependence-based design, dependent instructions are collected together, so instruction control logic within a cluster is likely to be simplified, because there is no need to look for independence if it is known not to exist. For example, if all the instructions assigned to a cluster are known to form a dependence chain (or nearly so), they can be issued in order from a FIFO, greatly simplifying instruction control logic.

The formation of dependent instructions can be done by the compiler, or at various stages in the pipeline, the dispatch stage being a good possibility. Waiting until the issue stage as in the 21264 may reduce inter-unit communication slightly, but at the expense of more complex issue logic.

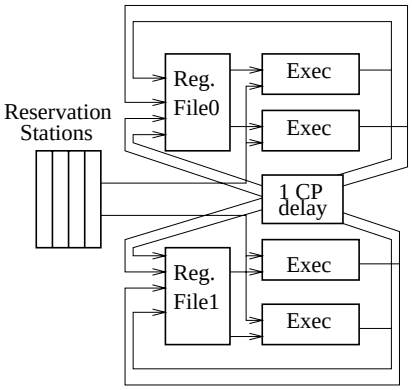

Fig. 3. Alpha 21264 clustered microarchitecture.

3.2. Heterogeneous ILDP

Another model for an ILDP is a heterogeneous system where a simple core pipeline is surrounded by outlying *helper* or *service* engines (Fig. 4). These helper engines are not in the critical processing path, so they have non-critical communication delays with respect to the main pipeline, and may even use slower transistors to reduce static power consumption.

Examples of helper engines include the pre-load engine of Roth and Sohi [9] where pointer chasing can be performed by a special processing unit. Another is the branch engine of Reinman et al. [10]. An even more advanced helper engine is the instruction co-processor described by Chou and Shen [11]. Helper engines have also been proposed for garbage collection [12] and dynamic correctness checking [13].

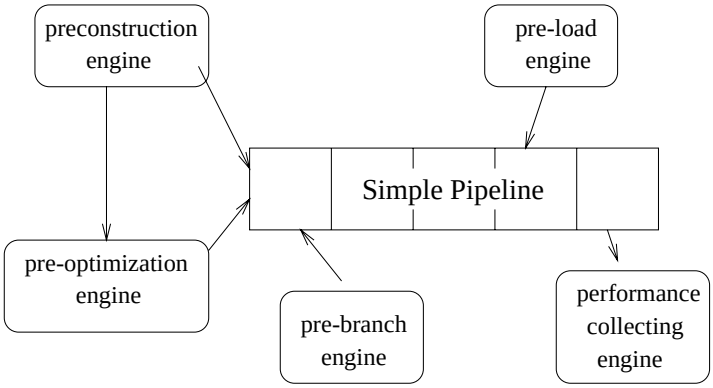

Fig. 4. A heterogeneous ILDP chip architecture.

4. Managing ILDP: Co-Designed Virtual Machines

It seems clear that an ILDP computer will need some type of higher level management of the distributed resources used by executing instructions. This management involves the steering of instructions and data among the units that compose the processor. It also involves allocation of the distributed resources for multiple simultaneous threads and power managment.

One option is for instruction interactions to be analyzed by compiler-level software. At compile time, inter-instruction dependences and communication can be determined (or predicted), then this information can be encoded into machine level instructions. At runtime this information can be used to steer instruction control and data information through the distributed processing elements.

Alternatively, hardware can be used to determine the necessary inter-instruction attributes by using hardware tables to collect dynamic history information as programs are executed. Then, this history information can be accessed by later instructions for steering of control and data information.

Overall management of processor resources is another important consideration; for example, resource load balancing will likely be needed for good performance -- at both the instruction level and thread level. For power efficiency, gating off unused or unneeded resources requires usage analysis and coordination, especially if power gating is widespread across a chip. This function can potentially be done via hardware or software, implemented as part of the OS.

Although they are viable solutions, a big disadvantage of software approaches based on conventional OS and compilers is that they likely require re-compilation and OS changes to fit each particular ILDP hardware platform. Disadvantages of the hardware-intensive solution are complex, power-consuming hardware and a rather limited scope for observing and collecting information related to the executing instruction stream.

A more radical and innovative solution is provided by currently evolving dynamic optimizing software and virtual machine technologies. A *Co-designed Virtual Machine* is a combination of hardware and software that implements a virtual architecture. The key idea is to provide hardware designers with a layer of software that resides in a hidden portion of DRAM main memory. This software layer will allow relatively complex dynamic program analysis and optimization. The base technology is used by Transmeta [14] and the IBM Daisy/BOA projects [15] primarily to support whole-system binary translation. The IBM 390 processors use a similiar technology, "millicode" [16], to support execution of complex instructions.

Fig. 5 illustrates the overall structure. The physical main memory space addressable by implementation hardware is larger than the memory space addressable programs supported by the architected hardware. Code and data to be used for managing ILDP hardware is placed in "hidden memory" at boot time.

At certain times during normal program execution (application or OS) the hardware can save the current program counter value and load the PC with a pointer into VMM code. The hardware may be designed to invoke the VMM in this manner at the time of (selected) program traps or systems call/returns, and special traps to the VMM can be initiated via a timer that the VMM controls.

After it takes control, the VMM saves architected state that it may modify. Then, it uses the processor in the normal fashion, fetching instructions from hidden memory, loading and storing data from its space in hidden memory. When it is finished, the VM returns control to the PC of the interrupted program and hardware resumes normal program execution.

In this system, all conventional software is completely off-the-shelf and no modifications have to be made in order for it to run correctly. In the meantime, hardware implementors are free write the VMM to optimize code for the ILDP implementation and to manage hardware resources (for either performance or power management [17]) "under the covers".

5. The Role of Instruction Sets

Historically, major changes in instruction set design have occured in discrete steps. After the original mainframe computers, instruction sets more-or-less stabilized until the early 1970s when minicomputers came on the scene. These machines used relatively inexpensive packaging and interconnections, and provided an opportunity to re-think instruction sets. Based on lessons learned from the relatively irregular mainframe instruction sets, regularity and "orthogonality" became the goals. To incorporate these properties, minicomputer ISAs typically supported relatively powerful, variable-length instructions; the PDP-11 and later VAX-11 instruction sets are good examples. As microprocessors evolved toward general purpose computing platforms, there was another re-thinking of instruction sets, this time with hardware simplicity as a goal. The resulting RISC instruction sets, among other things, allowed a full pipelined processor implementation to fit on a single chip. As transistor densities have increased, we have reached the point where older, more complex microprocessor instruction sets can now be dynamically translated by hardware into RISC-like operations.

In retrospect, it seems that instruction set innovation has keyed off packaging/technology changes, from mainframes to minicomputers to microprocessors. Now may be a good time to have another serious investigation of instruction sets. This time, however, on-chip communication delays, high speed clocks, and advances in translation/virtual machine software provide the motivation.

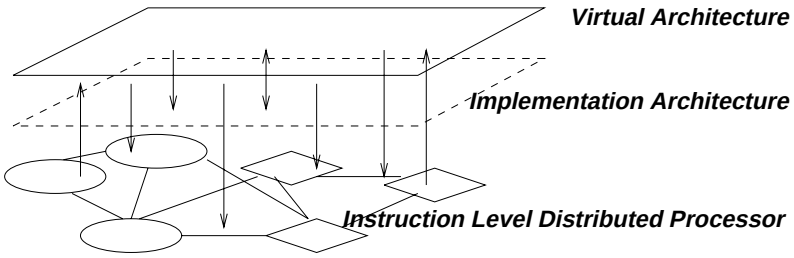

Fig. 5. Supporting an instruction level distributed processor with a co-designed Virtual Machine.

Instruction sets can and should be optimized for ILDP. Features of new instruction sets should focus on communication and dependence. They should also be optimized for very fast execution with emphasis on small, fast memory structures, including caches and registers.

Most recent instruction sets, including RISC instruction sets, and especially VLIW instruction sets, have emphasized computation and independence. The motivation was that higher parallelism could be achieved by focusing on computational aspects of instruction sets and on placing independent instructions in proximity either at compile time or during execution time. In contrast, ILDP instruction sets should be targeted at communication and dependence. That is, communication should be easily expressed and dependent instructions should be placed in proximity, to reduce communication delays.

Although maintaining compatibility with legacy program binaries has inhibited new instruction set architectures, virtual machine technology and binary translation enable new implementation-level instruction sets. That is, a translator/optimizer program can be placed in the hidden memory, and can be invoked by the VMM. Translated binaries can be cached in the hidden memory; this is basically the Transmeta/Daisy paradigm [refs], but applied to ILDP rather than a VLIW instruction set. Consequently, the complex optimizations required by VLIW are not necessary, and translation can be as simple as adding tag bits to provide instruction and data steering information.

To make ILDP instruction set concepts less abstract, consider an ISA incorporating a register file hierarchy by using a small fast register file for local communication within a cluster of processing elements, and a larger global register file for inter-cluster communication. Variable length instructions can be used to provide smaller instruction footprints and smaller caches.

As an extreme example, consider the following accumulator-based ISA. Assume 64 general purpose registers and a single accumulator are used for performing operations. All operations must involve the accumulator, so dependent operations are explicitly apparent as is local value communication. With such an ISA, there is need for only one general purpose register field per instruction and the ISA can be made quite compact, with instructions 1,2 or 4 bytes in length. For example, consider the following basic instruction types.

```
R <- A                  1 byte
A <- R                  1 byte
A <- A op R             2 bytes
A <- A op imm           4 bytes
A <- M(R op imm)        4 bytes
M(R op imm) <- A        4 bytes
R <- M(A op imm)        4 bytes
```

The first two instructions copy data to/from a register and the accumulator; A is the single accumulator and R is one of the general purpose registers. The next two instructions are examples of operations on data held in the accumulator (and general register file). The last three instructions are example loads and stores.

With an instruction set of this type, dependent instructions will naturally chain together via the accumulator and will be contiguous in the instruction stream. With a clustered ILDP implementation, all the instructions in a dependent chain can be steered simultaneously to the same cluster, with the next dependent chain being steered to another cluster. If the accumulator is re-named within each cluster, the parallelism among dependence chains can be exploited, with global communication taking place via the general registers. Because it contains only dependent instructions, the instruction issue queue in each cluster will be simplified as will local data communication through the accumulator.

The single accumulator ISAs is a simple example of an instruction set that implicitly specifies communication/dependence information; whether the implicit or explicit (tagging) methods are better is not clear. The important point is that instruction sets deserved renewed study, and future technologies and ILDP microarchitectures provide fertile ground for innovation.

6. Summary

Technology and application shifts are pointing toward instruction level distributed processing. These microarchitectures will contain distributed resources and will be explicitly structured for inter-unit communication. By constructing simple distributed processing elements, a very high clock rate can be achieved, probably with multiple clock domains. Replicated distribution processing elements will also allow better power management. Helper processors may be distributed around the main processing elements to perform more complex optimizations and to perform highly parallel tasks.

Virtual machines fit very nicely in this environment. In effect, hardware designers can be given a layer of software that can be used to coordinate the distributed hardware resources and perform dynamic optimization from a higher level perspective than is available to hardware alone. Finally, it is once again time that we reconsider instruction sets with the focus on communication and dependence. New instructions sets are needed to mesh with ILDP implementations, and they are enabled by the VM paradigm which makes legacy compatibility less important at the implementation architecture level.

Acknowledgements

This work was supported by National Science Foundation grant CCR-9900610, by IBM Corporation, Sun Microsystems, and Intel Corporation. This support is gratefully acknowledged.

References

1 V. Agarwal, M. S. Hrishikesh, S. W. Keckler, D. Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," *27th Int. Symp. on Computer Architecture,*, pp. 248-259, June 2000.

2 R. Eichemeyer, et al. "Evaluation of Multithreaded Uniprocessors for Commercial Application Environments", *23rd Annual Int. Symp. on Computer Architecture*, pp. 203-212, June 1996.

3 K. Diefendorff, "Compaq Chooses SMT for Alpha," *Microprocessor Report*, pp. 1, 6-11, Dec. 6, 1999.

4 L. Barroso, et al., "Prianha: A Scalable Architecture Based on Single-Chip Multiprocessing," *27th Int. Symp. on Computer Architecture,*, pp. 282-293, June 2000.

5 W. Yamamoto, M. Nemirovsky, "Increasing Superscalar Performance Through Multistreaming," 3rd Int. Symp. on Parallel Arch. and Compiler Techniques, June 1995.

6 D. Tullsen, S. Eggers and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", *22nd Annual International Symposium on Computer Architecture,* June 1995.

7 S. Palacharla, N. Jouppi, J. E. Smith, "Complexity-Effective Superscalar Processors," *24th Int. Symp. on Computer Architecture,*, pp. 206-218, June 1997.

8 L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report,* pp. 11-16, Oct. 1996.

9 A. Roth and G. Sohi, "Effective Jump-Pointer Prefetching for Linked Data Structures,"
26th Int. Symp. on Computer Architecture,, pp. 111-121, May 1999.

10 8. G. Reinman, T. Austin, B. Calder, "A Scalable Front-End Architecture for Fast Instruction Delivery," *26th Int. Symposium on Computer Architecture*, pp. 234-245, May 1999.

11 Yuan Chou and J. P. Shen, "Instruction Path Coprocessors," *27th Int. Symposium on Computer Architecture*, pp. 270-281, June 2000.

12 Timothy Heil and J. E. Smith, "Concurrent Garbage Collection Using Hardware-Assisted Profiling," *International Symposium on Memory Management (ISMM)*, October 2000.

13 T. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *32nd Int. Symposium on Microarchitecture*, pp. 196-297, Nov. 1999.

14 A. Klaiber, "The Technology Behind Crusoe Processors," *Transmeta Technical Brief*, 2000.

15 K. Ebcioglu and E. R. Altman, "DAISY: Dynamic Compilation for 100% Architecture Compatibility," *24th Int. Symp. on Computer Architecture,*, June 1997.

16 C. F. Webb and J. S. Liptay, "A High-Frequency Custom CMOS S/390 Microprocessor," IBM Journal of Research and Development, July 1997.

17 D. H. Albonesi, "The Inherent Energy Efficiency of Complexity-Adaptive Processors," *1998 Power-Driven Microarchitecture Workshop*, pp. 107-112, June 1998.