Microprocessors Head Toward MP on a Chip

Future Processors May Use Multiple CPUs to Boost Performance

by Linley Gwennap

*A recent story (see **080205.PDF**) examined the use of VLIW architectures to increase performance in future highly parallel microprocessors. This article discusses an alternative method of using increased parallelism to achieve high performance.*

Striving to improve performance, many computer scientists see a limit to simply adding more function units to a CPU. Some studies have indicated that there may not be enough instruction-level parallelism (ILP) in typical programs to make effective use of very many function units. If this limit is real, no instruction set—CISC, RISC, or VLIW (very long instruction word)—will see a significant benefit from processors that are much more complex than those that ship in the next year. Without alternative designs, performance improvement will be limited to increases in transistor speeds.

At last year's Microprocessor Forum, computer architect John Hennessy proposed placing multiple CPUs on a single chip to break this potential bottleneck *(see **071604.PDF**)*. By executing two or more instruction streams, such a chip could increase the available parallelism and thus use a large number of function units more effectively than a uniprocessor chip. This concept includes several possibilities, such as homogeneous MP, heterogeneous MP, and a shared-dispatch approach. Some of these techniques have already been implemented, and all are likely to appear in mainstream microprocessors by the end of this decade.

The Cookie-Cutter Approach

The simplest way to implement multiple processors on a chip is to place two identical processors onto a die. Given the high pin counts of most modern processors, however, it would be prohibitive to duplicate the system interface for two processors in a single package. Therefore, it would be best for the processors to share a single system interface, as Figure 1 shows. This interface must have enough bandwidth to handle the two processors.

Figure 1 shows two processors, each with its own individual caches. In this situation, the single system interface must check the bus transactions generated by each processor against the data cache of the other to maintain consistency. As in most MP systems, dual-ported cache tags reduce the overhead of this transaction snooping. An advantage of MP on a chip is that, on a snoop hit, wide on-chip buses could quickly update the other data cache, reducing MP overhead. This design could easily be extended to three or more processors.

One variation adds a unified second-level cache to the system interface. (This approach was used in a two-processor chip that Hitachi Research Labs presented at ISSCC in 1992 but never commercialized.) Given a fixed transistor budget, this variation requires the individual caches to be reduced in size. Despite the smaller primary caches, the overall hit rate will be improved, as the second-level cache can be dynamically allocated between the processors and between instructions and data.

This variation also eases data sharing and, as a side

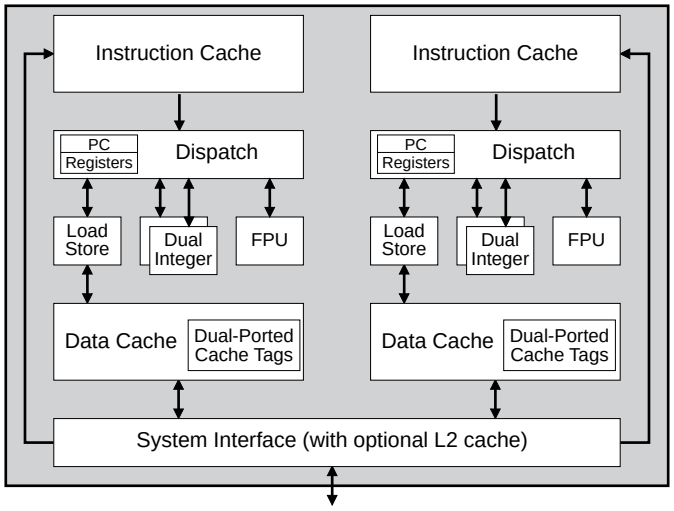

Figure 1. A chip with two processors that share the same system interface would be relatively easy to design.

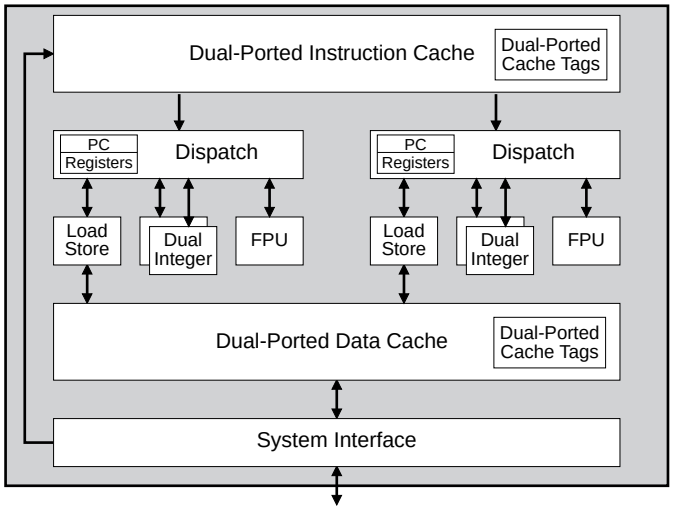

Figure 2. In this shared-cache design, a single set of instruction and data caches supports two processors.

effect, reduces the size of the dual-ported tag arrays. With the second-level cache on-chip, the first-level miss penalty should be only one cycle (assuming no contention), and a complete cache line can be refilled in a single cycle using a wide on-chip bus.

Figure 2 shows an alternative design that uses two large shared caches, one containing instructions for both processors and the other containing data. In this design, the caches must be dual-ported, perhaps using a multibank strategy, to avoid restricting execution speed to the access rate of the cache. A single data cache eliminates the need for on-chip snooping. As in the two-level cache design, it is easier for the processors to share data, and the processors can dynamically determine the amount of cache needed by each at any given time.

This design would be difficult to extend to more than two CPUs, because each CPU must be able to access the cache simultaneously. With four processors, for example, the cache tags would have to have four ports and the data arrays would need a large number of banks to avoid conflicts. Because of the linear nature of instruction accesses, a set of prefetch buffers could be used to reduce the number of accesses to the instruction cache; the data cache also could have less than one port per processor if the design could tolerate occasional conflicts. Both of these changes, however, increase complexity relative to the split-level cache approach.

One CPU, Multiple Program Counters

Figure 3 takes the concept of sharing to the extreme by using a unified instruction dispatcher. From the diagram, it is nearly impossible to distinguish this design from a classic superscalar processor. The trick lies in implementing multiple logical register files and multiple program counters. Instead of fetching eight instructions from a single instruction stream, this processor fetches four instructions each from two streams. Because the two streams use different logical registers, there can be no register dependencies between the streams. The dispatcher attempts to execute as many instructions as possible using a large pool of function units.

This chip contains a single physical register file divided into two logical register sets, one for each stream. The decoder simply adds a high-order bit to each register address to indicate which register set should be used; the function units then operate as if there were a single register file. As in a highly superscalar processor, the register file must be multiported, but the number of accesses to either half of the physical register file would be limited by the number of instructions that could be dispatched from a single stream (four in this example).

This approach, suggested by Digital's Dick Sites *(see **061606.PDF**)*, uses its function units more efficiently than other MP designs. For example, if one stream includes two consecutive floating-point instructions, those

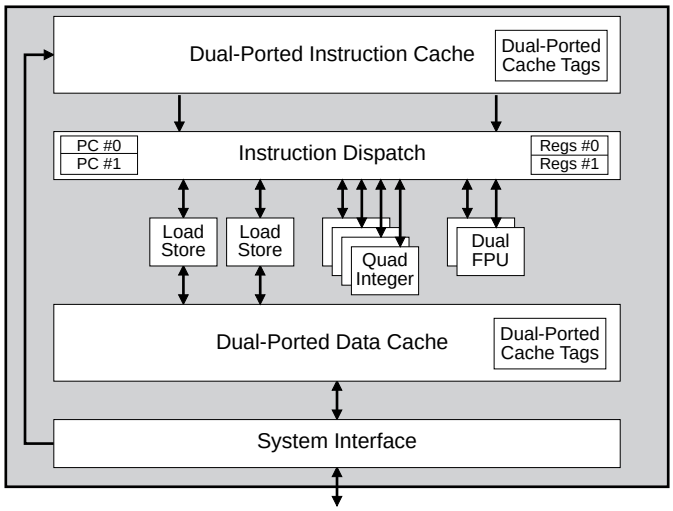

Figure 3. A more complex design fetches instructions from multiple streams and executes them using a single pool of function units.

instructions will take two cycles to execute on the shared-interface or shared-cache processors; the shared-dispatch design can execute them in a single cycle if both FPUs are available.

Because of this efficiency, the designer could repartition or reduce the number of function units. The processor in Figure 3 retains the same set of function units as the other example designs, but four integer units may be overkill; three could suffice. Certainly one integer multiplier would be adequate, whereas the other examples require two. Depending on the target application, one FPU might be enough. Thus, the area spent on the more complex dispatcher could probably be regained by eliminating excess function units, although the added complexity might also slow the clock.

Like the shared-cache processor, however, the shared-dispatch design needs complicated caches capable of sourcing data from several addresses at once. The ability of these caches to handle multiple accesses would be the factor limiting the number of instruction streams that could be handled by a single chip.

Easier to Design Than Complex CPUs

The shared-interface processor has the benefit of significantly reducing design complexity. This chip could simply combine existing processor designs. Only the system interface would have to be modified, and the complexities of cache coherency among multiple processors are well understood from discrete MP systems.

The other two options are complicated mainly by the cache design but offer the advantage of using on-chip resources more efficiently. Dual-ported caches are already appearing in Pentium-class x86 processors (due to the large number of memory references in x86 code) and will probably be used in some next-generation RISC chips; this effort could be leveraged in a future MP chip.

The Limits of Parallelism

Superscalar processors examine incoming instructions and attempt to issue two or more to the internal function units on each cycle. In theory, a processor with four function units could execute four instructions every cycle, delivering four times the performance of a scalar (single-issue) processor of the same clock speed.

In reality, however, it is nearly impossible for such a processor to sustain anything close to four instructions per cycle. Because each function unit can handle only certain types of instructions, some units will be idle frequently unless the ratio of instruction types matches the ratio of function-unit types. Some instructions cannot execute until a previous instruction calculates a result, often stalling one or more function units. Branches, which occur in typical programs every six instructions or so, cause problems by changing the flow of instructions in mutable and often unpredictable ways.

Several studies have attempted to determine the maximum amount of instruction-level parallelism (ILP) in programs (see MPR 9/19/90, p. 13). Although the results are not conclusive, some have indicated that, even with an infinite number of function units, ILP peaks at three to five operations per cycle for typical (nonscientific) code. Josh Fisher and other compiler gurus argue that new compiler techniques will increase ILP *(see **080205.PDF**)*, but the jury is still out.

Without new compiler techniques, it seems fruitless to build processors with more than six to eight function units. Several mainstream microprocessors with this number of function units are due in the next year, including the PowerPC 604 and 620, MIPS T5, Digital's 21164, and Sun's UltraSparc. These vendors could discover that the next step is a long and difficult one.

Even so, the shared-interface processor will have significantly lower design costs and, more important, a faster design cycle. As discussed, the shared-interface approach is also the easiest to extend to more than two processors, due to the complexity of the shared caches in the other designs. With individual primary caches, the only limit to placing multiple processors on a chip is the available transistor budget.

Any of the MP approaches would be simpler than designing a highly superscalar RISC (or CISC) CPU. Such a processor must incorporate highly accurate branch prediction, speculative and out-of-order execution, and complex dependency checking. Many of these features could be left out or simplified in MP chips, due to the lower issue rate of their CPUs. The complex design of a highly superscalar processor, combined with the increased probability of design bugs, could significantly extend the design cycle of these chips. In the fast-moving computer market, long design cycles may be the fatal flaw of complex superscalar processors.

Combining Different Processors

Multiple processors on a chip are not constrained to homogeneous arrangements. For at least some applications, it is advantageous to specialize the processors. In the simplest case, one could contain a floating-point unit while the others do not; FP-intensive instruction streams could be routed to that processor by the operating system. This design would eliminate the cost of implementing a full FPU in one or more processors.

Special-purpose hardware also could be added to one of the processors to create a heterogeneous design. For example, a fast integer multiplier would speed signal-processing algorithms. A Huffman decoder and DCT (discrete cosine transform) engine accelerate video decompression. MP chips will probably migrate toward heterogeneous designs to take advantage of such features without burdening all processors on the chip.

In the extreme case, the processors need not share a compatible instruction set. Texas Instruments' MVP *(see **080405.PDF**)* includes four symmetric DSPs plus a RISC CPU, all on a single chip. The RISC processor uses a different instruction set than the DSPs, making the RISC portion easier to program, particularly for simple overhead tasks. The DSPs, on the other hand, offer better number-crunching performance but typically must be hand-coded.

Another example is Motorola's 68322 processor for printers *(see **080604.PDF**)*. This $18 chip combines a small 68000 CPU core with a specialized graphics processor. The two autonomous processors work in parallel to increase throughput; restricting the function of the second processor to a single task improves its performance. This design uses its processing power more efficiently than a faster uniprocessor chip. Zilog has taken a similar tack, combining a low-cost Z8 processor with a DSP on a single chip, as have a few other manufacturers.

The downside of a heterogeneous design lies in matching the resource mix to the target application. For the MVP, the ratio of four DSPs to one RISC CPU makes it suited for high-end signal processing. A workstation, on the other hand, would be better served by a chip with four RISC CPUs and one DSP. With the wrong mix, portions of the design will be poorly utilized.

Software Must Be Rewritten

A VLIW processor *(see **080205.PDF**)* would be easier to design than any comparable MP chip. Because it delivers pregrouped operations in a single long instruction, VLIW eliminates the dispatch logic and does not require a multiported instruction cache. Ideally, it could also achieve better performance than the MP designs by using the compiler to most efficiently schedule the function units. The simplicity of the VLIW hardware might also allow it to reach faster clock speeds.

This performance boost, however, assumes that new compilers will be able to expose enough ILP for the VLIW chip to operate effectively; current compilers do not achieve this feat. Any VLIW processor would also require a new instruction set; at best, it could emulate existing RISC or CISC binaries. The MP designs have the advantage of compatibility with existing instruction sets and current compiler technology.

MP designs require their own leap of faith, however. To take advantage of multiple processors, applications must be rewritten with multiple threads.

Threading exposes task-level (coarse-grained) parallelism, as opposed to the instruction-level parallelism sought by superscalar and VLIW designs. For example, a payroll program might generate a hundred paychecks. Instead of executing these hundred tasks serially on a single processor, each paycheck can be assigned to a different thread; these threads can then be distributed among any number of processors to increase throughput. Because the task of generating one paycheck does not affect any other, these tasks can occur in parallel.

For the most part, compilers cannot identify tasks that are intuitively obvious to humans. Thus, threading is typically done by hand; programmers must rewrite applications to use multiple threads. Most commercially available software, particularly for PCs, is not multithreaded and must be rewritten to take advantage of future MP designs. Otherwise, its performance will be no better than on single-processor systems.

One advantage of desktop software for MP is that an increasing percentage of time is spent in the user interface. Future interfaces will take advantage of 3-D graphics, video decompression, speech synthesis, and voice recognition, all tasks that are relatively easy to thread. Furthermore, if common interface routines are threaded, they can be leveraged among many applications to increase performance on MP systems even if the core application code is not threaded.

Is the World Ready for Multiprocessing?

There is little difference between MP on a chip and a discrete MP system from the perspective of the user, the application, or even the operating system. Although MP systems have found some success in specialized areas such as scientific computing and high-end servers, they have not done well in volume markets, particularly on the desktop. This could lead to the conclusion that MP chips would have a limited market as well.

The biggest problem with MP systems today is the lack of MP software. Most versions of Unix now support multiple processors, but mainstream operating systems such as Windows, Macintosh, and NetWare do not. Chicago, the next generation of Windows, will be multithreaded but will not support multiple processors. Either Windows NT, which already supports MP, or a future MP version of Chicago will bring multiprocessing to the masses; other OS vendors will probably follow suit.

ISVs have been slow to develop multithreaded applications due to the effort required and the lack of MP operating systems. The release of Chicago, which is expected to ship tens of millions of copies in 1995, will spur ISVs to thread their applications. As full MP operating systems become more popular, they will further encourage application vendors to make this effort.

The other piece of the puzzle is the availability of MP hardware. Intel is acting to ensure the widespread availability of low-cost MP systems within the next few years *(see **080603.PDF**)*. This may not be a coincidence: Intel's CISC architecture constrains its ability to create highly superscalar x86 processors, and the advantage of software compatibility may lead that company down the path to multiple processors on a chip.

A Plethora of Design Options

Looking out to 1997, we expect transistor budgets for high-end microprocessors to reach 10 million or more, giving chip designers several options. The current path leads to a 12-way superscalar processor (three times the complexity of a PowerPC 604) with 128K of cache on chip. This would be a very difficult design to get to market, and it may not achieve much better performance than a processor with an eight-issue design.

Another option would be to build an eight-issue processor and expand the on-chip cache from 128K to 256K. For many applications, however, such an increase would not significantly increase the cache hit rate.

VLIW is a third option. By eliminating the dispatch complexity, a 10-million-transistor VLIW chip might hold 16 or more function units and 128K of cache. If compiler technology advances enough to keep this vast number of function units busy, the VLIW chip could outperform the superscalar designs.

The same transistor budget could be devoted to three 604-like processors, each with 16K of cache, plus a shared interface with a 64K second-level cache. This chip would have lower uniprocessor performance than the others, but by 1997 a large base of multithreaded applications should exist. For these programs, the MP chip should deliver similar or better performance than the superscalar chip, depending on the degree of threading in each program. The MP chip would also be much easier to design, reaching the market sooner than the others.

Once a base of MP operating systems and multithreaded software exists, MP chips become feasible. Even if compiler technology can expose enough ILP for highly superscalar and VLIW processors, MP chips could be good midrange options. Without these magic compilers, MP chips could be a desirable solution for many markets. Given the three-year microprocessor design cycle, the time is right to begin exploring these options. ♦

