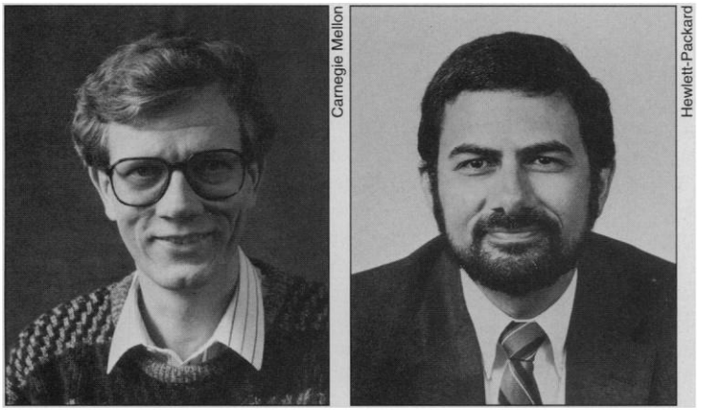
A Painless Route to Parallel Computing?

Specialized processors are ferreting out and exploiting the parallelism in conventional programs

SCIENTISTS AND OTHER COMPUTER users lusting after ever greater quantities of affordable number-crunching power have long been seduced by the prospect of inexpensive supercomputers fashioned from hundreds or thousands of microprocessors churning away in parallel. But the charms of cheap parallelism still have to be paid for—with new software. Code written for single-processor machines must be rewritten to divvy up computing tasks among multiple microprocessors, and that can be a monumental task. Some workers are now touting a solution: Instead of yoking together multiple processors to form a parallel computer, why not build a single microprocessor that acts like a parallel computer but uses conventional code?



Sizing up the gains. *Thomas Gross* (left) *tempers Joseph Fisher's optimism about ILP's potential to speed up computing.*

That's the vision behind instruction-level parallelism (ILP), an approach described in this issue of *Science* by Joseph Fisher and Ramakrishna Rau of the Hewlett-Packard Laboratories in Palo Alto (see p. 1233). In ILP, a single, specially designed processor simultaneously runs different tiny chunks of a standard program. Simple forms of ILP have in fact been boosting the speed of some supercomputers and mainframes for 30 years. But in the past few years, computer designers have begun to push the idea to new levels of performance and apply it to the microprocessors that power a much wider range of computers, including the workstations that are a mainstay of scientific computing.

Already, a few computer makers have turned out new chips that owe some of their blinding speed to ILP, and some ILP researchers—including Fisher and Rau—insist that ILP-driven speedups of 20-fold or more are just around the corner. Other computer scientists aren't so sure, arguing that most programs simply don't lend themselves to being broken up into tiny parallel pieces. Besides, they say, even if ILP works as advertised, it can't compete with the benefits of a large-scale move to parallel computers, in the form either of stand-alone machines or of networks of workstations.

What nobody disputes is the vexing inefficiency of conventional, "serial" computing. A conventional microprocessor executes a program's instructions one at a time in an order strictly dictated by the program. To add two numbers, for example, the processor must first pull one number from the computer's memory, then pull the second number, then add the two, then store the result in memory. Since any single instruction rarely calls the chip's full capabilities into play, the result of this numbingly sequential approach is that most of its resources stand idle most of the time.

ILP puts some of this wasted power to work by modifying chips to carry out two or more instructions at once. Thus an ILP-based chip told to add two numbers might pull both of them from memory at the same time rather than one after the other. It couldn't execute the "add" instruction in parallel with the memory calls, of course, because it wouldn't know which numbers to add. But if the chip still had capacity to spare, it might get to work on another instruction from further down the line in the program—one that doesn't depend on the results of any as-yet-unexecuted operations.

That instruction might be one that is executed every time the program is run—but it doesn't have to be. Computer programs are full of branchpoints, where they split into two or more paths, only one of which is followed each time the program is run. For example, a program may call for adding two numbers, then executing one set of instructions if the sum is less than 6 and another if the sum is 6 or more. Even before it knows the result of the addition, an ILP chip can go ahead and "speculatively" follow either of the paths; if it later turns out that the chip took the wrong path, it can simply discard the results. It's a gamble, but one that can pay off handsomely in speedup. For that matter, a sufficiently powerful ILP chip can cover its bases by speculatively executing both branches.

Indeed, designing a chip that can run multiple instructions at once is the easy part. The real cost of ILP, in complexity and processing overhead, comes in identifying instructions that are independent of earlier results—hence free to be executed ahead of time. In some ILP schemes this chore falls to the compiler, the specialized program that translates the "high-level" language of the software into the language of the chip. As it does so, the compiler can search out interdependencies and flag them for the chip. In another approach, the chip itself looks ahead as it works, scanning upcoming code for instructions that can safely be pulled down for early execution.

An even simpler ILP strategy known as pipelining has been speeding up many minicomputers and mainframes since the early 1960s. In pipelining, the processor and compiler look no further ahead than the next instruction; if that instruction doesn't depend on the result of the ongoing one, the processor starts work on it early. By overlapping the execution of two or more successive instructions, pipelining alone can double the speed of a processor.

Now entering the market are ILP chips—among them the microprocessors made by MIPS and Sun and the chips in several IBM and Intel workstations and minisupercomputers—that exploit more sophisticated variants of ILP to achieve greater gains. And to hear some researchers tell it, that's just for starters. "For scientific applications, speedups of factors of 20 or more won't be unusual at all," says Fisher. "It's still a mystery what to expect from ILP for other types of software, but I wouldn't be all that surprised if it turned out to be 8 or 10 for system software [operating systems] and as much as 20 for many commercial applications, such as payroll processing."

But other researchers say ILP is likely to max out well short of those numbers. The barrier: the confounding interdependency of instructions in a typical program. "Most

studies show that ILP can bring a maximum overall improvement of 2 to 8 times, and even that may be difficult to obtain," says Thomas Gross of Carnegie-Mellon University who has helped design ILP-based processors. "The people who claim there is more parallelism than that available simply haven't been able to prove it."

The trouble with proving anything about ILP, Gross notes, is the lack of a universally accepted benchmark for ILP processor performance. For tasks such as image processing, where the same instructions are carried out independently on many separate chunks of data, he says, ILP is already achieving the kind of speedups Fisher is talking about. But Gross suspects that for applications lacking such repetition, such as word processing, ILP could even be slower than standard sequential computing, owing to the added calculational overhead.

In any case, Gross asserts that efforts to wring the last ounce of parallelism from existing programs by ILP may be wasted in view of the much greater potential of true parallel computing. "Even if you were able to get a 20-times speed improvement over a wide range of programs," he says, "it would still be small compared with the improvement you could get with multiprocessor parallel computing."

David Gelernter, a Yale University computer scientist who researches ways of linking workstations in parallel networks, echoes that assessment. ILP should be seen not as an alternative to general purpose parallel computing but as an adjunct to it, he says. "If ILP can make the workstations in a network faster, that's great," he says. "But if you're talking about going with ILP instead of a network, forget it. In the final analysis the fastest machine will always be 2 or 100 or N of something tied together." As for the difficulties of rewriting conventional software for large-scale parallelism, Gelernter claims they have been greatly exaggerated.

Fisher is unfazed by such indifference to ILP. He thinks Gelernter and others are underestimating the challenge of converting code to parallel networks. And even if that obstacle is cleared away earlier than he expects, he thinks it's silly to worry that hopes for ILP will keep people from pursuing full-scale parallel processing. For a user with access only to a single workstation, he points out, ILP is the only road to parallelism. Worrying about a conflict, he says, "is a little like saying a car that corners better is dangerous because it might keep people from taking an airplane when they want to get across the country." ■ DAVID H. FREEDMAN

David H. Freedman is a contributing editor of Discover *magazine.*

Hydroxyl, the Cleanser That Thrives on Dirt

Pollution may have reduced this atmospheric cleanser, but half-measures against pollution might make matters worse

THE HYDROXYL RADICAL IS THE PAC MAN OF Earth's atmosphere. Brought to life by a zap of solar radiation, this molecule of hydrogen and oxygen spends the second or so before it flickers out of existence scooting about gobbling up most anything that has been fouling the air—carbon monoxide that leads to smog, methane that enhances the greenhouse, sulfurous gases, and unburned oil. By oxidizing and thus eliminating these contaminants, the voracious hydroxyl serves as the mainstay of the global atmosphere's self-cleansing process, holding at bay noxious gases produced by natural processes and, more recently, doing its best to mitigate the worst excesses of human activity. It's no surprise, then, that hydroxyl has probably lost some of its zip of late.

In the face of the mounting load of atmospheric pollutants, the molecular supercleanser's lifetime seems to have gotten even shorter, with little compensating speedup in its primary source—an ultraviolet-driven reaction between ozone and water. As a result, hydroxyl is less abundant now than when the industrial era began, some scientists suspect. Not that they have been able to measure the abundance of the fleeting molecule directly (see box). But their recent computer modeling of the changing chemistry of the lowermost layer of the atmosphere, called the troposphere, suggests that since 1700 hydroxyl has decreased by perhaps 5% to 20%.

All is not lost, however. When the computer models that paint this discouraging picture are extended into the future, they offer a little encouragement—laden with a lot of irony. As even more pollutants accumulate, the models suggest, the atmosphere could shift into a new mode of operation that produces hydroxyl faster, bolstering the air's self-cleansing ability. But the boost in hydroxyl would come from some of the most noxious pollutants themselves—ones that are causing acid rain, destroying the ozone layer, and possibly warming the globe. Well-intentioned efforts to control these contaminants could condemn hydroxyl to a continuing decline, making the atmosphere even less able to cope with other pollutants. To scientists, the only clear lesson seems to be that halfway attempts to clean up the atmosphere will likely not be enough; the offending chemicals are too tightly linked in an intricate web of chemical and physical interactions.

The modelers who are bringing this mes-

Pinning Down a Will-o'-the-Wisp

For 20 years the hydroxyl radical has thumbed its nose at atmospheric chemists. The molecule, consisting of an oxygen bound to a hydrogen, is a pivotal player in the maze of chemical reactions that determines the composition of the lower atmosphere, but a practical means of measuring it has long eluded researchers. At a concentration of one hydroxyl for every 10 trillion air molecules, it is simply too scarce to be measured directly with sufficient speed and precision. But with theoretical models suggesting that this key atmospheric cleanser has suffered a worrisome decline (see main text), atmospheric chemists are more eager than ever to develop measuring devices.

Now several researchers think they have hydroxyl in their sights. The current leaders in the hydroxyl hunt are atmospheric chemists Fred L. Eisele and David Tanner of the Georgia Institute of Technology, who stumbled on a means of measuring it one summer day in 1987 as they were monitoring atmospheric ions near a high-voltage DC power line in Massachusetts. They were searching for ions produced by the power line, but they couldn't help noticing one species that was clearly responding to something else: the bisulfate ion, which peaked at midday and plummeted every time a cloud passed in front of the sun.

After a couple of days' thought, Eisele and Tanner realized what was happening. They knew that bisulfate ions are produced when hydroxyl radicals oxidize sulfur dioxide, a common pollutant. The rise and fall of bisulfate, they concluded, had to be tracking the rise and fall of hydroxyl as clouds and the passage of the sun altered the supply of