Forth Report

Editor: *Paul Frenger, P.O. Box 820506, Houston, TX 77282; pfrenger@ix.netcom.com*

Parallel Forth

Paul Frenger

1 Introduction: Cranking Up the Old Central Processor

The majority of digital computers used on this planet run their programs sequentially. That is, they fetch an instruction from memory, decode it, then execute the operations embodied in the nstruction. This simple cycle is repeated on and on until the omputer is halted. This kind of computer is sometimes described as a Von Neuman machine, after John Louis von Neumann (1903-1957), a mathematician and promoter of the stored computer program concept [1]. It's a great design, robust and effective. However, its sequential architecture acts a lot like the escapement mechanism in a clock, holding back central processor speed in a tick-tock gait. This has been referred to as the 'Von Neuman bottleneck' [2]. Since people usually don't like to wait, engineers and computer scientists have improved, enhanced and honed this little sequential computing machine continuously over the years using a variety of techniques. One stratagem has been to raise the CPU clock speed; we started with a 4.77 MHz Intel 8088 in the IBM PC in 1983, and now we are enjoying 450 MHz Pentium IIs, with faster chips on the near horizon. Raising CPU speed is a non-trivial task, however, since extensive changes to chip geometry, transistor technology, semiconductor fabrication methods and power management have had to be made. Some of these CPU chips work near the limits of heat tolerance; occasionally one oversteps these bounds in a smoky, sparking display. Even printed circuit board fabrication technology has had difficulty keeping up with these super-fast CPUs.

Another CPU performance-enhancing stratagem has been widening of on-chip buses and word widths. Our 1983 PC used an 8-bit CPU, quickly going to 16-bits, then 32 and 64-bits. These new wide-bus chips can take in and spit out more bytes of information from memory (as instructions or data) in a single cycle. Unfortunately, wider buses commit more of the chip's internal silicon acreage to signal conduction rather than to computational units.

Other stratagems have been used to further enhance the modern central processor's speed and general performance: pipelining technology, provision for large on-chip caches, scoreboarding, out-of-order instruction execution, math coprocessors, having multiple execution units and so on. Technologies not-yet implemented (such as three-dimensional architectures) are waiting in the wings. But ultimately, the bottlenecks in Von Neuman's machine must assert their performance limitations, just as the speed of light in a vacuum affects space travel (the 'Star Trek' legend notwithstanding).

Can we circumvent this execution speed problem somehow?

2 Introducing Parallelism, With Some Forth

When cheap or simple is the criterion, one of something usually suffices. For example, years ago Motorola sold a single-bit CMOS CPU, the MC14500B, which they called the 'Bit Fiddler' [3]. Hardly more than an ALU, this 1 MHz 16-pin IC had no address bus, recognized sixteen 4-bit opcodes (two of which were no-ops!) and performed arithmetic and boolean functions on one data bit at a time. You configured a MC14500B system by connecting a free-running up-counter to the address lines of a 4-bit wide program EPROM, and the four EPROM data lines fed the CPU its instructions. When the counter was reset to zero the first instruction was fetched from the EPROM, then executed by the MC14500B. Each instruction in turn was fetched as the counters incremented. When the counters overflowed to zero again, the program restarted. If you were clever with CMOS 4000-series and 74C-series 'glue chips', you could create quite complex and sophisticated systems with it. Motorola intended the part to be a super-simple processor for such applications as traffic light sequencers, but single-chip microcontrollers were a heckuva lot easier to use, and one of today's Microchip Technologies' PIC 12C-series 8-pin 8-bit OTP controllers would run rings around it. So, no, I'm not surprised you never heard of the MC14500B.

While on the subject of single-bits and CPUs, Motorola also created a cheap version of its 8-bit 6800 processor architecture, the 6804, which (if I heard this right) used a single-bit internal serial bus to move data bytes around on the chip. This serial approach was about eight times slower than using traditional 8-bit parallel data paths, but did meet the criterion of saving expensive silicon for transistors since it required only one signal conductor internally. But with silicon geometries now shrinking to nearly nothing and microcontrollers hitting 49 cents in quantity (as a recent Zilog ad bragged), there is no justification for a slow cheap chip when fast cheap chips are available.

I had a teacher in school who used to justify group homework assignments by saying "many hands make light work". Well they do, but the resulting interpersonal confusion can be a killer. In our discussion today, I'm going to present a few examples of this 'many hands' approach for your amusement and edification.

A good example of parallelism was shown to the attendees of the 1991 SIGForth Conference in San Antonio, Texas [4]. This Dr Howerton had a student in his programming languages class, a Joseph Gradecki, who built a parallel computer using sixteen Intel 8031 8-bit microcontrollers arranged in a hypercube. Each 8031 'node' contained a home-grown Forth interpreter; each communicated serially with its nearest neighbor nodes and with a master CPU which controlled the entire hypercube. Joseph created a simple parallel-OS based on Forth. He demonstrated this device by calculating Mandelbrot sets, performing parallel text searches and pipelined computation of polynomials. With an 8031 internal clock speed of 3 MHz, the 'cube' achieved 192 MIPS and between 4 and 8 megaFLOPS. This is excellent performance, especially for such dinky little 8-bit controllers, ten years ago! And again, using today's microcontrollers (like the Scenix 50 MHz version of the Microchip Technology PIC 16C5x), you could build a big, fast Forth hypercube on a single Eurocard board.

Heck, even I got into this act, and at that same ACM SIGForth Conference mentioned above [5]. I designed a multi-layer neural network system using New Micros' NMIS-0021 Forth-based Motorola 68HC11 single board computers [6]. I created an 'A-neuron' and a 'B-neuron' (which differed only in whether the Forth software used the 68HC11's analog or digital I/O) on each 2-inch by 4-inch CPU board. Each board simulated 8 hybrid neurons, which were directly connected vertically and horizontally to other neurons in the various layers. The simplest 3-layer 8-neuron neural network consisted of two New Micros' boards, connected together to form a 4-inch square, one-half-inch tall. Networks containing more layers and/or more neurons could be assembled using more boards, all controlled by the Forth language. Obviously, this is a different kind of parallelism than that mentioned above (artificial analog neural network vs. digital hypercube), but I think this example illustrates the principle okay.

Next we're going to jump past 8-bit processor power, bursting into the realm of 16-bit to 32-bit node networks and well beyond.

3 More Parallelism, More Forth

In the early 1980's, now-gone INMOS created their first Transputer CPU, the T-212 [3]. It was a 16-bit stack/accumulator design with a RISC-like 'minimized' instruction set. What made the Transputer ideal for parallel processing networks was its built-in hardware supporting fast CPU-to-CPU communications. Each Transputer chip had four high-speed serial data links with on-chip RAM buffers and DMA processing, so that interprocessor communications would take place in a background mode, not requiring constant CPU supervision.

INMOS developed the Transputer through 32-bit versions (T-414 and T-800) and a 64-bit bus version (the T-9000). The T-800 had an internal 64-bit floating point ALU, somewhat reminiscent of the Intel 486. The T-9000 also added a 50 MHz clock, much faster processing speed, a 16 kbyte cache, more hardware communications support and a 5-stage instruction pipeline. Instructions were bytecoded (4-bits for commands and 4-bits for data or workspace offsets), so a single 64-bit memory access could pick up eight opcodes at a time which the CPU could execute in parallel. This particular Transputer was ideal for parallel systems known as 'systolic arrays'. The T-9000, combined with C104 crossbar switches, could be used in networks containing hundreds of thousands of processors. That would be a BIG network!

The operating system INMOS supplied for the Transputer was called 'occam' [7], written by David May [8] and named after the English philosopher William of Occam (1300-1349). Occam was designed to easily describe concurrent processes which communicate via one-way channels. The 'process' is the essential operation in occam, with four subtypes: 'assignment', 'input', 'output' and 'wait'. These subtypes could be used to construct more complex processes for sequential or parallel execution, which could use inputs from other CPUs. Conditional execution was supported. A revised version, 'occam2', added floating-point math and data typing [9].

Well, having such a powerful stack-oriented CPU at hand, how long do you think Forth enthusiasts and Forth language suppliers could resist writing a suitable Forth for the Transputer? Answer: not very long at all. One Transputer enthusiast [10] has listed a number of programming languages which have been implemented on Transputer networks; several Forths are included, some of which are mentioned below.

The first is 'Inmos Transputer Forth', which is found in the parallel computing archive at the University of Kent in England [11]. The listings given cover Forths from the 16-bit T-212 to the 32-bit T-800 Transputers. The necessary hardware is described, Forth images and a loader is provided, and instructions on rebuilding the Forths for other configurations are given.

Next is a Forth-79 for the Transputer. Noel Henson [12], a former team leader at Cogent Research, designed a desktop computer using 32 INMOS T-414 and T-800 processors. He reported in a posting to comp.lang.forth on May 4, 1998 that the only quick way to debug the hardware was to use Forth. Consequently he and his team wrote a Forth-79 for this hardware, which executed quickly on the Transputer and which could replicate itself from node to node. The program files archived in 'rom.zip' can be downloaded through links at Mr. Meenakshisundaram's website.

Next is 'Turbo-Forth 83' for the 32-bit T80x Transputers. The program files archived in 'f-tp-100.zip' can be downloaded through the above website links. Finally there is 'tForth Parallel Forth v1.0' by The Dutch Forth Workshop, accessible via the above website or directly [13]. This transputer Forth contains an integrated optimizer and an assembler; it runs about half as fast as hand-optimized assembler but much faster than Transputer C code. It is ANS-compliant. The I/O of tForth is handled with a server written in the C language and is currently available for MS-DOS and UNIX machines.

The above Transputer Forths may be satisfactory for general use, but I have not had any personal experience with them. My sainted grandmother always used to say, "If you don't know diamonds, know your jeweler". To stretch a metaphor a bit, let me rephrase that wisdom: "If you don't know Forth, know your Forth source". One Forth vendor I know quite well who offers Transputer Forth is MicroProcessor Engineering Ltd of Southampton, England. They sell a PC-based or Unix-based Forth cross-compiler for the T2xx, T4xx and T8xx CPUs [14].

The above list is undoubtedly incomplete. Be sure to check the relevant newsgroups [15-17], the Internet Parallel Computing Archive [11], the Oxford University Transputer Archive [18] and try keyword searches with your favorite search engines. By the way, the Transputer was picked up commercially from INMOS by STMicroelectronics (formerly SGS-Thomson) [19]. Please contact them for pricing and availability.

Finally in this section, let me mention the use of Forth with the legendary Massively Parallel Processor (MPP) developed by Dr. John Dorband, NASA/Goddard Space Flight Center in the mid-1980's [20]. The MPP consisted of 16,000 individual single-instruction multiple-data (SIMD) processors, arranged in clusters. Each of these was fed by its own instruction stream. Communication between clusters of processors was accomplished both explicitly and implicitly. The MPP was used for image enhancement from spacecraft and the Hubble Telescope, complex graphics operations, simulation of cellular automata, finite element analysis and solution of other amazingly complex problems. It is unlikely that you or I will ever get our hands on such a vast parallel computer, but if I did I would use it to analyze DNA structure, protein synthesis and enzyme binding site dynamics. What would you use it for?

Dorband has worked on a number of other parallel computers. He has been a member of the ANS Forth Standards Committee, which proves that he knows a good thing when he sees it.

4 Latest Parallel Forth Developments

Some of the preceeding information pertains to the use of Forth on ancient or unobtainable hardware. Now let's look at current events with Jeff Fox, a long-time Forth enthusiast, hardware/software designer and parallel computing wizard [21]. Jeff has worked with Forth creator Chuck Moore for a decade, including several Forth-in-silicon projects.

Fox has looked long and hard at marrying Forth with LINDA, which is not a programming language but a way of extending conventional languages into the parallel processing arena (ie: FORTRAN-LINDA, C-LINDA, FORTH-LINDA) [22]. The LINDA paradigm is based on active and passive 'tuples'. A tuple is a sequence of typed fields each of which have (or can have) a value. Tuples exist in 'tuple space', which is a global, associative, object memory. It can be implemented on multiprocessor systems or computer networks. An active tuple supports processes which are executing; passive tuples exchange information (like 'Post-It' notes). Active tuples can spawn simultaneous 'child' processes in other tuples as needed to perform calculations in parallel; passive tuples synchronize these processes and move data around.

In FORTH-LINDA, the master processor manages the network, breaks up large problems into smaller parts and manages the active 'worker tuples' which perform the necessary calculations in parallel. When a worker tuple finishes its current task, it is recycled and reassigned to a new task. The number of available processors in the network is irrelevant: the more CPUs which are available to run active tuples, the faster the problem can be solved.

The master processor loads a copy of Forth into each available processor. At the lowest level, these Forth processors execute Forth code pretty much in a normal fashion, with the usual stack operations being local to the processor. At a higher level, LINDA extension words to Forth can be executed by these same processors, reading or writing data and/or messages in passive tuples. FORTH-LINDA has inherent load balancing. The Montvelishsky paper describes all this in great detail and it is beautiful to read and understand.

Obviously, any networked computer system which could run Forth could support FORTH-LINDA, from that little 8031-based hypercube, to the Transputers, to the MPP. But Jeff Fox has a cute little CPU for us which further enhances the beauty of this approach: a Forth engine known as the 'F21' [23]. The F21 is a stand-alone 20-bit Forth processor which can be networked in rings, grids, multidimensional arrays or other network topologies as needed to support various types of problems. As a stand-alone CPU it contains analog A/D-D/A, parallel ports, a realtime clock, a video coprocessor for generating TV-type synchronized NTSC or RGB color displays, a memory controller (SRAM/DRAM/ROM) and high-speed DMA-supervised serial I/O for networking. A CPU cycle is 20 nsec.

Why is the F21 a 20-bit processor? Essentially, because this is all it needs. Each of the CPU's 27 instructions is 5-bits long, allowing four instructions to be fetched per memory cycle and executed simultaneously. A maximum sustained rate of 200 MIPS executing out of SRAM can be achieved in this fashion (ROM is faster, DRAM slower). Jeff Fox offers a free F21 software simulator and emulator which you can download from his website. The F21 chip is now up to revision 'd' and is packaged in a 68-pin PLCC. Please contact Jeff at Ultra Technology for pricing and availability.

Fox and Montvelishsky worked together on 'F*F', their Forth-distilled version of LINDA for the F21. The original FORTH-LINDA implementation required five Forth words to accomplish this; the new F*F requires only two, capitalizing on the built-in communications hardware support of the F21 when used in networks. The passive tuple model has been removed from LINDA, vastly simplifying data communications and message-passing between active processes. Any F21 can send data to a single node, groups of nodes or to the entire network; the Forth word 'G!' stores data to these global data structures. Reading global variables or data structures work like local memory fetches and are quite fast. The Forth word 'RX' sets up a Forth word execution stream for transmission to and running on other nodes. This F*F software could be run on other hardware if you'd like to homebrew a few boards with your favorite CPU and some glue logic; however, the F21 seems to be the ideal setup for F*F, especially since its instruction set is truly made for Forth.

5 Conclusion

This long-winded odyssey has gone backward and forward in time, looked at serial and parallel CPUs, singly and by the hundreds of thousands, and hardware and software considerations for parallel processing. The good news is: by now, your understanding of parallel processing and the role of Forth hardware and software in it is considerably greater. The bad news is: we have barely scratched the surface of this topic! I invite you to look more deeply if you have an interest in this area, or better still have a proposed application. Contact the people and resources I have outlined for you. Get involved!

So there you have it: when you speak of parallel computing systems, don't forget to think of Forth.

6 Bibliography

1. http:// ei.cs.vt.edu/ ~history/ VonNeumann.html.

2. Howland, John, "Lecture Notes for CSCI 301: Great Ideas in Computer Science", Dept Comp Sci, Trinity Univ, San Antonio, TX, 1995. http:// www.cs.trinity.edu/ About/ The_Courses/ cs301/ 12.parallel.processing/ 12.parallel.proc.html.

3. http:// www.glue.umd.edu/ ~dikmen/ CPUs.html.

4. Howerton, Charles, "Forth is alive and well and living in a hypercube somewhere in Wyoming", Proc ACM SIGForth'91 Conf, pg.115-119.

5. Frenger, Paul, "A Forth-Based Hybrid Neuron for Neural Nets", Proc ACM SIGForth'91 Conf, pg.99-102.

6. http:// www.newmicros.com.

7. http:// www.realtime-info.be/ encyc/ techno/ terms/ 87/ 64.htm.

8. May, David, "Occam", Sigplan Notices, Vol.18 No.4, 1983, pg.69-79.

9. http:// www.realtime-info.be/ encyc/ techno/ terms/ 88/ 64.htm.

10. Meenakshisundaram, Ram, "Transputer Home Page". http:// www.geocities.com/ SiliconValley/ Heights/ 1190/ languages.htm.

11. http:// www.hensa.ac.uk/ ftp/ pub/ parallel/ vendors/ inmos/ archive-server/ forth/ .

12. http:// www.cowboyz.com.

13. http:// www.iaehv.nl/ users/ mhx/ t4faq.html.

14. http:// www.mpetest.demon.co.uk/ MPE.htm.

15. comp.parallel.

16. comp.sys.transputer.

17. comp.lang.forth.

18. http:// www.comlab.ox.ac.uk/ archive/ transputer/ .

19. http:// www.st.com.

20. http:// newton.gsfc.nasa.gov/aCe/ aCe_dir/ C_language/ references/ PForth.html.

21. http:// www.ultratechnology.com.

22. Montvelishsky, Michael, "PARALLEL FORTH: The New Approach", 1993. http:// www.ultratechnology.com/ 4thpar.html.

23. Fox, Jeff, "F21 and F*F: F21 and Parallizing Forth", 1993. http:// www.ultratechnology.com/ forml93.html.

Paul Frenger is a medical doctor who has been professionally involved with computers since 1976. He has worked as a computer consultant, published nearly one hundred articles in the bioengineering and computer literature, edited the ACM SIGForth Newsletter for four years and acquired three computer patents along the way. Paul was bitten by the reverse Polish bug in 1981 and has used Forth ever since. Being both a physician and a computer programmer, Paul believes that the term 'hacker' is doubly appropriate in his case.