The Tera Computer System*

Robert Alverson David Callahan Daniel Cummings Brian Koblenz
Allan Porterfield Burton Smith

Tera Computer Company
Seattle, Washington USA

1 Introduction

The Tera architecture was designed with several major goals in mind. First, it needed to be suitable for very high speed implementations, *i.e.*, admit a short clock period and be scalable to many processor sets. This goal will be achieved; a maximum configuration of the first implementation of the architecture will have 256 processors, 512 memory units, 256 I/O cache units, 256 I/O processors, and 4096 interconnection network nodes and a clock period less than 3 nanoseconds. The abstract architecture is scalable essentially without limit (although a particular implementation is not). The only requirement is that the number of instruction streams increase more rapidly than the number of physical processors. Although this means that speedup is sublinear in the number of instruction streams it can still increase linearly with the number of physical processors. The price/performance ratio of the system is unmatched, and puts Tera's high performance within economic reach.

Second, it was important that the architecture be applicable to a wide spectrum of problems. Programs that do not vectorize well, perhaps because of a preponderance of scalar operations or too-frequent conditional branches, will execute efficiently as long as there is sufficient parallelism to keep the processors busy. Virtually any parallelism available in the total computational workload can be turned into speed, from operation level parallelism within program basic blocks to multiuser time- and space-sharing. The architecture even has strong support for implementing non languages like Lisp and Prolog and highly ap languages like Sisal and Id.

A third goal was ease of compiler implementa Although the instruction set does have a few un tures, these do not seem to pose unduly hard p for the code generator. There are no register addressing constraints and only three addres Condition code setting is consistent and orth though the richness of the instruction set o several ways to do something, the variation i tive costs as the execution environment chang be small. Because the architecture permits t change of spatial and temporal locality for a high-quality optimizing compiler may work hard i ing locality and trade the parallelism there more speed. On the other hand, if there is s parallelism the compiler has a relatively ea

The Tera architecture is derived from that o [9, 10], although they are highly similar m MIMD systems, there are many significant diff between the two designs.

2 Interconnection Network

The interconnection network is a three-di mesh of pipelined packet-switching nodes, ea is linked to some of its neighbors. Each link port a packet containing source and destin dresses, an operation, and 64 data bits in b tions simultaneously on every clock tick. S nodes are also linked to *resources*, *i.e.*, processor memory units, I/O processors, and I/O cache u stead of locating the processors on one side work and memories on the other (in what Robert has called a "dancehall" configuration[5]), t are distributed more-or-less uniformly thr network. This permits data to be placed in units near the appropriate processor when tha ble and otherwise generally maximizes the di tween possibly interfering resources.

*This research was supported by the United States Defense Advanced Research Projects Agency under Contract MDA972-89-C-0002. The views and conclusions contained in this document are those of Tera Computer Company and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. Government.

The interconnection network of a 256 processor Tera system contains 4096 nodes arranged in a 16×16×16 toroidal mesh; that is, the mesh “wraps around” in all three dimensions. Of the 4096 nodes, 1280 are attached to the resources comprising 256 processors, 512 data memory units, 256 I/O cache units and 256 I/O processors. The 2816 remaining nodes do not have resources attached but still provide message bandwidth. To increase node performance, some of the links are missing. If the three directions are named X, Y, and Z, then X-links and Y-links are missing on alternate Z layers. This reduces the node degree from 6 to 4, or from 7 to 5 counting the resource link.

In spite of its missing links, the bandwidth of the network is very high. Any plane bisecting the network crosses at least 256 links, giving the network a data bisection bandwidth of one 64-bit data word per processor per tick in each direction. This bandwidth is needed to support shared memory addressing in the event that all 256 processors are addressing memory on the other side of some bisecting plane simultaneously.

As the Tera architecture scales to larger numbers of processors p, the number of network nodes grows as $p^{3/2}$ rather than the $p\log(p)$ associated with the more commonly used multistage networks. For example, a 1024-processor system would have 32,768 nodes. The reason for the overhead per processor of $p^{1/2}$ instead of $\log(p)$ stems from the fact that the system is speed-of-light limited. One can argue that memory latency is fully masked by parallelism only when the number of messages being routed by the network is at least $p\times l$, where l is the (round-trip) latency. Since messages occupy volume, the network must have volume proportional to $p\times l$; since the speed of light is finite, the volume is also proportional to l^3 and therefore l is proportional to $p^{1/2}$ rather than $\log(p)$.

3 Memory

A full-sized system contains 512 data memory units of 128 megabytes each. Memory is byte-addressable, and is organized in 64-bit words. Four additional *access state* bits, more fully described in section 5, are associated with each word. Data and access state are each equipped with a separate set of single error correcting, double error detecting code bits. Data addresses are randomized in the processors using a scheme similar to that developed for the RP3[8]. The randomization is excellent for avoiding memory bank hotspots and network congestion, but makes it difficult to exploit memory locality using nearby memory units. In the Tera system, the randomization is combined with another notion called *distribution*. The processor data segment map has a distribution factor associated wit ment. Consecutive virtual addresses in are distributed among all 512 data memory a single unit, or any power of two in between.

Disk speeds have not kept pace with advances sor and memory performance in recent yea only currently reasonable solution to this p lower the level of disks in the memory hierarc large semiconductor memory between th and data memory. In a fully configured Tera s that 70 gigabyte per second sustained bandwid between secondary storage and data memory is s by 256 I/O cache units comprising a directly addressable memory of 256 gigabytes.

The I/O cache units are functionally iden data memory. The only difference is that thei is higher because their memory chips are sl denser). The fact that I/O cache has all of the functionality of memory makes it possible to map I/O b directly into the address spaces of the appl grams that access them. This is used to avoi by remapping segments.

A processor fetches instructions through the nearest neighboring I/O cache unit. This av work traffic and network latency, but requires of the program be made for every processor it i

4 Processors

Each processor in a Tera computer can execute instruction streams simultaneously. In the plementation, as few as one or as many as 128 p counters may be active at once. On every tic clock, the processor logic selects a stream t to execute and allows it to issue its next i Since instruction interpretation is complet by the processor and by the network and memor well, a new instruction from a different strea issued in each tick without interfering with When an instruction finishes, the strea belongs thereby becomes ready to execute instruction. As long as there are enough i streams in the processor so that the average i latency is filled with instructions from oth the processor is being fully utilized. Thus necessary to have enough streams to hide the latency (perhaps 70 ticks on average); once hidden the processor is running at peak per and additional streams do not speed the resul If a stream were not allowed to issue its nex until the previous instruction complet approximately 70 different streams would be rec

each processor to hide the expected latency. The look-head described in section 4.3 allows streams to issue multiple instructions in parallel, thereby reducing the number of streams needed to achieve peak performance.

4.1 Stream State

Each stream has the following state associated with it:

- 1 64-bit Stream Status Word (SSW)
- 32 64-bit General Registers (R0-R31)
- 8 64-bit Target Registers (T0-T7)

Context switching is so rapid that the processor has no time to swap the processor-resident stream state. Instead, it has 128 of everything, *i.e.*, 128 SSW's, 4096 general registers, and 1024 target registers. It is appropriate to compare these registers in both quantity and function to vector registers or words of cache in other architectures. In all three cases, the objective is to improve locality and avoid reloading data.

Program addresses are 32 bits long. Each stream's current program counter is located in the low half of its SSW. The upper half describes various modes (*e.g.* floating point rounding, lookahead disable), the trap disable mask (*e.g.* data alignment, floating overflow), and the four most recently generated condition codes. Most operations have a `_TEST` variant which emits a condition code, and branch operations can examine any subset of the last four condition codes emitted and branch appropriately.

Also associated with each stream are 32 64-bit general registers. Register `R0` is special in that it reads as 0, and output to it is discarded. Otherwise, all general registers are identical.

The target registers are used as branch targets. The format of the target registers is identical to that of the SSW, though most control transfer operations only use the low 32 bits to determine a new PC. Separating the determination of the branch target address from the decision to branch allows the hardware to prefetch instructions at the branch targets, thus avoiding delay when the branch decision is made. Using target registers also makes branch operations smaller, resulting in tighter loops. There are also skip operations, which obviate the need to set targets for short forward branches.

One target register (`T0`) points to the trap handler, which is nominally an unprivileged program. When a trap occurs, the effect is as if a coroutine call to `T0` had been executed. This makes trap handling extremely lightweight and independent of the operating system. Trap handlers can be changed by the user to achieve specific trap capabilities and priorities without loss of efficiency.

4.2 Horizontal Instructions

Processor effectiveness, the utilization of t interpretation resources, has always been co the difficulty of issuing more than one instru tick. This difficulty has become known as the Bottleneck[2]. Vector instructions sidest culty in part, but are not able to handle freq tional branches or heterogeneous scalar oper Processors with horizontal instructions, ex ples of which are sometimes called Very Long tion Word (VLIW) architectures, offer a good tive to vector instructions. In a horizontal several operations are specified together. M erations are usually simple loads and store others are two- or three-address register-t erations. If the overall architecture and are capable of achieving one instruction per every functional unit mentioned in the inst well-used. If the instructions are only mode branches can be sufficiently frequent.

Tera instructions are mildly horizontal. cally specify three operations: a memory ref eration like `UNS_LOADB`(yte), an arithmetic op like `FLOAT_ADD_MUL`(tiply), and a control operat `JUMP`. The control operation can also be a secon metic operation, `FLOAT_ADD`, or perhaps an `INTEGER` used in an address computation. Vectorizab can be processed at nominal vector rates (on tick) using only horizontal instructions wit kinds of operations. Matrix-vector multipl tains nearly two flops per tick via the same t used for its efficient vectorization.

4.3 Explicit-Dependence Lookahead

If there are enough streams executing on each to hide the average latency (about 70 ticks) th chine is running at peak performance. Howeve stream can execute some of its instructions (*e.g.* 2 successive loads) then fewer streams an activities are required to achieve peak perf The obvious solution to this problem is to i instruction lookahead; the only difficulty is The traditional register reservation a quires far too much scoreboard bandwidth in t of architecture. Either multi-streaming or instructions alone would preclude scoreboar The traditional alternative, exposing the pipe impractical because multi-streaming and unp memory operation latency make it impossible rate code that is both efficient and safe.

The Tera architecture uses a new techniqu explicit-dependence lookahead. The idea is

ple: each instruction contains a three bit lookahead field that explicitly specifies how many instructions from this stream will issue before encountering an instruction that depends on the current one. Since seven is the maximum possible lookahead value, at most eight instructions and twenty-four operations can be concurrently executing from each stream. A stream is ready to issue a new instruction when all instructions with lookahead values referring to the new instruction have completed. Thus, if each stream maintains a lookahead of seven, then nine streams are needed to hide 72 ticks of latency.

Lookahead across one or more branch operations is handled by specifying the minimum of all distances involved. The variant branch operations JUMP_OFTEN and JUMP_SELDOM, for high- and low-probability branches respectively, facilitate optimization by providing a barrier to lookahead along the less likely path. There are also SKIP_OFTEN and SKIP_SELDOM operations. The overall approach is philosophically similar to exposed-pipeline lookahead except that the quanta are instructions, not ticks.

4.4 Protection Domains

Each processor supports as many as 16 active protection domains that define the program memory, data memory, and number of streams allocated to the computations using that processor. Each executing stream is assigned to a protection domain, but which domain (or which processor, for that matter) is not known to the user program. In this sense, a protection domain is a virtual processor and may be moved from one physical processor to another.

The protection domains share a single 64K data segment map and a 16K program page map. Each protection domain has two pairs of map base and limit registers that describe the region of each map available to it. The upper 2048 data segments and 1024 program pages are not relocated by the map bases, and are used by the operating system. Any active protection domain can use all of either or both maps. The map entries contain the physical address; the levels of privilege needed to read, write, or execute the segment or page; whether the segment or page was read, written, or executed, as appropriate; and the distribution (for the data map).

The number of streams available to a program is regulated by three quantities slim, scur, and sres associated with each protection domain. The current number of streams executing in the protection domain is recorded by scur; it is incremented when a stream is created and decremented when a stream quits. A create can only succeed when the incremented scur does not exceed sres, the number of streams reserved in the protection domain. The operations for reserving streams

are unlimited, and allow several streams to be reserved simultaneously. The str slim is an operating system limit on the nu streams the protection domain can reserve.

When a stream executes a CREATE operation to c a new stream, it increments scur, generates th SSW for the new stream using one of its own target copies the trap target T0 from its own T0 regis loads three registers in the new stream fro general purpose registers. The newly creat can quickly begin executing useful work in co with its creator as long as significant stor tion is unnecessary. The QUIT operation termi stream that executes it, and decrements both scur. The QUIT_PRESERVE operation only decremen scur, thereby giving up a stream without surre its reservation.

Each protection domain has a *retry limit* that de mines how many times a memory reference can f testing a location's full/empty bit (see sec it will trap. If a synchronization is not sa long time, then possibly a heavier weight m that avoids busy waiting should be used to wai synchronization. The retry limit should be the amount of trap processing overhead, whi depending on the run-time environment. The t thus can invoke the heavier weight mechan appropriate.

4.5 Privilege Levels

The privilege levels apply to each stream inde There are four levels of privilege: user, su nel, and IPL. IPL level operates in absolute mode and is the highest privilege level. Use sor, and kernel levels use the program and d address translation, and represent incr of privilege. The data map entries define the levels needed to read and write each segment program map entries define the *exact* level need execute from each page. The current privile stream is stored as part of the privileged s and is not available to a user-level stream.

Two hardware operations are provided to al executing stream to change its privilege l (LEVEL_ENTER *lev*) operation sets the current priv level to the instruction map level if the cur equal to *lev*. The LEVEL_ENTER operation is locate every entry point that can accept a call from privilege level. A trap occurs if the curren equal to *lev*. The (LEVEL_RETURN *lev*) operation is use return to the original privilege level. A if *lev* is greater than the current privilege le

4.6 Exceptions

Exceptional conditions can occur in two ways. First, an instruction may not be executed due to insufficient privilege, as with a `LEVEL_RETURN` which attempts to raise the privilege level. This type of exception is quite easy to handle. More commonly, exceptions occur *while* executing instructions. With lookahead, further instructions may already be executing and overwriting registers which would be needed to restart instructions.

Rather than keep shadow copies of registers to support rollback, the Tera architecture defines certain exceptions as a side effect of instruction *completion*. In this model, exceptions are guaranteed to be signaled before they are needed, as indicated by the lookahead field. Thus, if instruction j depends on instruction i, all possible exceptions during the execution of instruction i will be signaled before instruction j begins execution.

To support diagnosis and recovery, certain state must be available to the trap handler. A trap can be caused by any of the three operations in an instruction each of the (at most eight) memory operations that trapped, the processor provides the trap handler with the trap reason and enough state to allow the operation to be retried (*e.g.* for demand paged virtual memory).

For arithmetic traps caused by the arithmetic operations no state is automatically provided to the trap handler. The decision to preserve operand values for possible use by the trap handler is made by the compiler. While the lookahead field normally only guards true dependence for registers, operand values may be preserved by limiting lookahead to guard antidependence as well.

5 Tagged Memory

Each memory location in the Tera computer system is equipped with four access state bits in addition to a 64-bit value. These access state bits allow the hardware to implement several useful modifications to the usual semantics of memory reference. The two data trap bits generate application-specific lightweight traps, the forward bit implements invisible indirect addressing, and the full/empty bit is used for lightweight synchronization. The influence of these access state bits can be suppressed by a corresponding set of bits in the pointer value used to access the memory.

The two trap bits in the access state are independent of each other and are available for use by the language implementer. If a trap bit is set in a location and the corresponding trap disable bit in the pointer is clear, a trap will occur. Uses for the trap bits include data breakpoints, demand-driven evaluation, run-time exception signaling, implementation of "active" memory objects, and even stack limit checking.

The forward bit implements a kind of "invisi rection". Unlike normal indirection, forwar trolled by both the pointer and the location If the forward bit is set in the memory locati warding is not disabled in the pointer, the va the location is to be interpreted as a pointe get of the memory reference rather than the ta Dereferencing will continue until the pointe ables forwarding or discovers that the addres has its forward bit reset. The primary use of is for on-the-fly modification of address-loc ings, for example in concurrent storage recl volving the copying of live structures from o another.

The full/empty bit controls the synchronizi ior of memory references. Load and store op can optionally use the full/empty bit in the memory word by setting bits in the access cont The four values for access control are shown

value	LOAD	STORE
0	read regardless	write regardless and set full
1	reserved	reserved
2	wait for full and leave full	wait for full and leave full
3	wait for full and set empty	wait for empty and set full

When access control is 2, loads and stores the memory cell to be full before proceeding context, it is sometimes useful to think of t as meaning "available" and the empty state as "unavailable". The reading or writing of any object is conveniently prevented by marking it as "unavailable". The access control values 1 and 3 cause loads to be treated as "consume" operations a as "produce" operations. A load waits for ful sets to empty as it reads, and a store waits for that sets to full as it writes. A forwarded loc is not disabled and that has its full/empty empty is treated as "unavailable" until it irrespective of access control.

Additional operations exist to fetch the a of a given memory location or to set the acces a given location.

Although the full/empty bit provides a fas of implementing arbitrary indivisible memory o the need for extremely brief mutual exclusi "integer add to memory" is so important for sc applications that this function is done ent each memory unit by a single operation, FETC This is the Ultracomputer fetch-and-add ope and differs from it only in that the network h

does not combine fetch-and-add operations to the same memory location.

6 Arithmetic

The numeric data types directly supported by the TERA architecture include:

- 64 bit twos complement integers
- 64 bit unsigned integers
- 64 bit floating point numbers
- 64 bit complex numbers

Operations on these types include addition, subtraction, multiplication, conversion, and comparison. Reciprocation of unsigned and floating point quantities is provided for using Newton's method.

Other types are supported indirectly, including:

- 8, 16, and 32 bit twos complement integers
- 8, 16, and 32 bit unsigned integers
- arbitrary length unsigned integers
- 32 bit floating point numbers
- 128 bit "doubled precision" numbers

The shorter integers are sign- or zero-extended to 64 bits quantities as they are loaded from memory, and truncated to the appropriate length as they are stored. The fundamental support for arbitrary length integer arithmetic is provided by the operations `INTEGER_ADD_MUL`, `UPPER_ADD_MUL`, and `CARRY_ADD_TEST` that together implement $64 \times n$ bit unsigned multiply-add in approximately $2\frac{2}{3}n$ instructions.

The 32 bit floating point numbers are simply the real parts of the 64 bit complex type with imaginary parts set to zero. The 128 bit "doubled precision" type was pointed out to us by Kahan [1, 7, 4]; it represents a real number R as the unevaluated sum of two 64 bit floating point numbers r and ρ, where ρ is insignificant with respect to r and as near as possible to $R-r$. Support for this type is provided by `FLOAT_ADD_LOWER` which (with `FLOAT_ADD`) implements "doubled precision" addition in six instructions, and by `FLOAT_MUL_ADD` which rounds only once and is used to implement "doubled precision" multiplication in five instructions.

References

[1] T. J. Dekker. A floating-point techniqu
tending the available precision. *Numrische Ma*
18:224–242, 1971.

[2] M. Flynn. Some computer organizations a
effectiveness. *IEEE Transactions on Computers*,
C-21(9):948–960, September 1972.

[3] A. Gottlieb, R. Grishman, C. P. Kruskal
McAuliffe, L. Rudolph, and M. Snir. The NYU
tracomputer - designing an MIMD shared me
parallel computer. *IEEE Transactions on Comput-*
ers, C-32(2):175–189, 1984.

[4] W. Kahan. Doubled-precision IEEE stand
floating-point arithmetic. Unpublished ma
February 1987.

[5] R. M. Keller. Rediflow: A proposed archit
combining reduction and dataflow. In *PAW83*
suls Used at the 1983 Parallel Achitecture Work-
shop, University of Colorado, Boulder, 1983

[6] J. T. Kuehn and B. J. Smith. The Horizo
percomputer system: Architecture and so
In *Proceedings of Supercomputing '88*, Orlando,
Florida, November 1988.

[7] S. Linnainmaa. Software for doubled-
floating-point computations. *ACM Transactions on*
Mthematical Software, 7:272–283, 1981.

[8] A. Norton and E. Melton. A class of bool
ear transformations for conflict-free po
stride access. In *Proceedings of the 1987 Inter-*
national Conference on Parallel Processing, pages
247–254, August 1987.

[9] Franka Pittelli and David Smitley. Anal
3d toriodal network for a sihared memory
tecture. In *Proceedings of Supercomputing '88*, Or-
lando, Florida, November 1988.

[10] M. R. Thistle and B. J. Smith. A process
tecture for Horizon. In *Proceedings of Supercomput-*
ing '88, pages 35–41, Orlando, Florida, Nove
1988.