A Microbenchmark for Characterizing Network IO on the Tera Multithreaded Architecture

CSE 260 Project #13
Greg Johnson
Dr. Jay Boisseau

Introduction

This project involves characterizing the performance of the Tera’s network IO. Specifically:

• system bottlenecks affecting network IO will be identified

• factors such as the number of threads required to "fill" the effective bandwidth of a standard 10 or 100 BaseT network connection will be determined
Tera MTA Macro Architecture

- Processor module
- CPU
- ~260 MHz
- 780 Mflops
- 2.1 GBps, 150 cycles
- Remote memory
- 1–4 GB per module

The Tera MTA installed at SDSC is currently the only one of its kind. The MTA is configured with 8 processors and 8 GB RAM, to be upgraded next week to 16 processors / 16 GB RAM.

* All numbers shown are peak.
* Bandwidth and latency shown where known.

Tera MTA Micro Architecture

- MTA processors are custom chips implemented in GaAs (though Tera is moving to silicon)
- the MTA works around the von–Neumann bottleneck by tolerating latency to memory vs. relying on cache or expensive high speed memory
- the MTA memory hierarchy is flat, avoiding data locality issues / cache level optimization, and simplifying load balancing
- the MTA design utilizes inexpensive commodity RAM
Tera MTA Micro Architecture

- threads are not bound to a given processor, and threads from the same program may run on different processors
- parallelism is limited only by system memory size and the amount of parallelism inherent in the target program
- overhead associated with creation of a thread is minimal
- threads from the same program share one address space
- synchronization between threads is fast due to hardware support, and while synchronization may stall one or more threads, the processor(s) running the threads will not stall
Tera MTA Programming Model

- taking advantage of parallelism on the Tera MTA relies *only* on identifying independent threads of execution within a program

```c
/* go thru the image a pixel at a time */
#pragma tera assert parallel
#pragma tera dynamic schedule
for ( index=0 ; index<xsize*ysize ; index++ ) {
    #pragma tera assert local my_ray
    my_ray = InitializeRay(my_ray, index);
    image[index] = SampleRay(my_ray, data_volume);
}
```

- the Tera compiler can find implicit parallelism within a program in cases such as loops with no data dependencies or indirect references to data elements

- the Tera compiler generates a report that details what parallelism it found in the code, along with what areas could not be automatically parallelized and why
Network IO Microbenchmark

ASSUMPTIONS

• one or more bottlenecks exist on the MTA which prevent it from driving a socket at rates seen on other platforms

• bottlenecks may be either disk or network interface related

• IO performance will improve as Tera refines their design

PURPOSE

• a microbenchmark will allow us to pinpoint the problem(s) to help direct Tera’s development efforts, and allow SDSC to chronical improvements to the system

GOALS

• *micromove* will be a multithreaded file transfer program for moving files of arbitrary size between the MTA and another host

• the *micromove* design will allow disk IO and network IO to be measured independently

• the *micromove* disk IO and network IO components will be multithreaded

• *micromove* will be run with files of different sizes across varying numbers of threads, resulting in a table showing transfer rates for file sizes vs. thread counts
Network IO Microbenchmark

This work is supported in part by NSF award ASC–9613855 and DARPA contract DABT63–97–C–0028.

Additional Information

- [http://www.tera.com/](http://www.tera.com/)
  The home page of Tera Computer Company.

  A page with links to many technical papers and slide sets regarding the MTA design, performance, compiler, and tools.

- [http://www.sdsc.edu/~allans/papers.html](http://www.sdsc.edu/~allans/papers.html)
  Allan Snavely’s collection of technical papers regarding the performance of the MTA on several benchmarks and real world problems.