Toward a Software Infrastructure for the Cyclops-64 Cellular Architecture

Juan del Cuvillo Weirong Zhu Ziang Hu Guang R. Gao
Department of Electrical and Computer Engineering
University of Delaware
Newark, Delaware 19716, U.S.A
{jcuvillo,weirong,hu,ggao}@capsl.udel.edu

Abstract

This paper presents the initial design of the Cyclops-64 (C64) system software infrastructure and tools under development as a joint effort between IBM T.J. Watson Research Center, ETI Inc. and the University of Delaware. The C64 system is the latest version of the Cyclops cellular architecture that consists of a large number of compute nodes each employs a multiprocessor-on-a-chip architecture with 160 hardware thread units. The first version of the C64 system software has been developed and is now under evaluation. The current version of the C64 software infrastructure includes a C64 toolchain (compiler, linker, functionally accurate simulator, runtime thread library, etc.) and other tools for system control (system initialization, diagnostics and recovery, job scheduler, program launching, etc.)

This paper focuses on the following aspects of the C64 system software: (1) the C64 software toolchain; (2) the C64 Thread Virtual Machine (C64 TVM) with emphasis on TiNy Threads™, the implementation of the C64 TVM; (3) the system software for host control. In addition, we illustrate, through two case studies, what an application developer can expect from the C64 architecture as well as some advantages of this architecture, in particular, how it provides a cost-effective solution. A C64 chip's performance varies across different applications from 5 to 35 times faster than common off-the-self microprocessors.

1. Introduction

The C64 is an ambitious supercomputer project currently under development at IBM Research Laboratory, which plans to deliver the world's first petaflop system by 2007. C64 is designed to serve as a dedicated compute engine for running high performance computing intensive applications. A C64 supercomputer is attached to a host system through a number of Gigabit Ethernet links. The host system provides a familiar computing environment to applications software developers and end users. Besides access (through the Ethernet links) to a common file server, each C64 chip can be connected to a serial ATA disk drive.

A C64 is built out of tens of thousands of C64 processing nodes. Each processing node consists of a C64 chip, external DRAM, and a small amount of external interface logic. A C64 chip employs a multiprocessor-on-a-chip architecture containing 160 hardware thread units, half as many floating point units, on-chip SRAM, on-chip instruction cache, bidirectional inter-chip routing ports, and interface to off-chip DDR SDRAM. On-chip resources are connected to a crossbar network, which also provides thread units access to the routing ports that connect each C64 chip to its neighbors arranged in a 3D-mesh configuration.

In this paper we present a system software architecture, encompassing components running on the host system and on C64 nodes, for system management and application development and execution. In particular, we focus on the following aspects of the C64 system software:

- The C64 Thread Virtual Machine (C64 TVM) and its realization in the form of the TiNy Threads™runtime library. We outline the three key components of the C64 TVM: the thread model, where thread management issues are presented; the memory model that includes a presentation on both C64 memory address space and memory consistency model; and the synchronization model that provides the functionality to implement mutual-exclusion regions, perform direct thread-to-thread and barrier-type of synchronizations. In addition, we discuss how TNT takes advantage of C64 specific hardware features to leverage performance without imposing any additional burden on the application programmer.

- The C64 software toolchain with emphasis on the sup-

port for segmented memory spaces, which is required to fully exploit the multi-level memory hierarchy that is directly visible by the programmer.

- The C64 management system developed to maximize the utilization of the C64 computing engine. On one hand, procedures to bootup nodes, verify that all systems are working properly, and recovering from errors are aimed to reduce the impact of hardware failures and minimize downtime. On the other hand, a job scheduler and launcher focus of preventing C64 nodes from being idle while jobs are waiting for resources.

Although performance tuning and optimization are not the objectives of this paper, we also report some initial performance observations to demonstrate what a software developer can expect to obtain from the C64 architecture. In particular we demonstrate that for some well known applications a C64 chip delivers the same performance as dozens of common off-the-self microprocessors.

2. Cyclops-64 cellular architecture

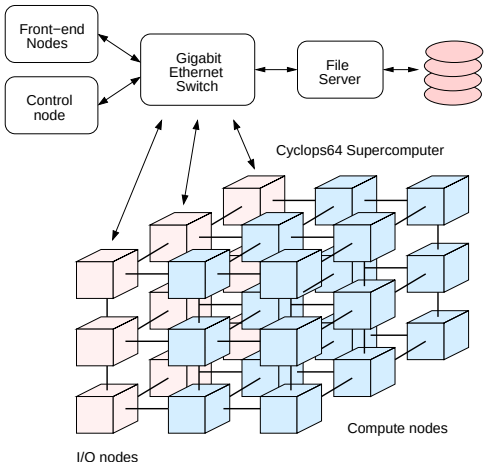

Figure 1. Cyclops-64 computing environment

The computing environment we are considering consists of a host and external file systems connected to a C64 supercomputer by means of a Gigabit Ethernet network, see Figure 1. The host system (shown as consisting of a number of control nodes and a front-end node) supports application program development and execution, as well as system administration. The file system, which may also contain multiple (external) file server nodes, provides one means of file support for the C64 supercomputer. An internal high bandwidth distributed file system hosted by serial ATA hard drives attached to each C64 node will also be available to avoid disk bottlenecks and network congestion.

C64 nodes are arranged in a 3D-mesh network. A fraction of these nodes, labeled as I/O nodes, use the Gigabit Ethernet port (present in all C64 chips) to connect the C64 supercomputer to the host and external file systems. Each I/O node will service a number of C64 nodes, called compute nodes, and relay requests and data between the compute nodes and the host and file server systems. The I/O nodes and compute nodes communicate via packets over the 3D-mesh network only. This 3D-mesh provides the high bandwidth necessary for internode communication in running application programs.

There is a separate control network that connects the C64 system to the host system. This control network carries commands from the host system to each C64 node. A C64 node attaches to this control network via its JTAG interface. The host system uses this control network to initialize the C64 system, monitor its status while programs are in execution, and reconfigure and restart C64 after hardware failures. Details of the initialization and configuration procedures are not the focus of this paper and will be discussed elsewhere.

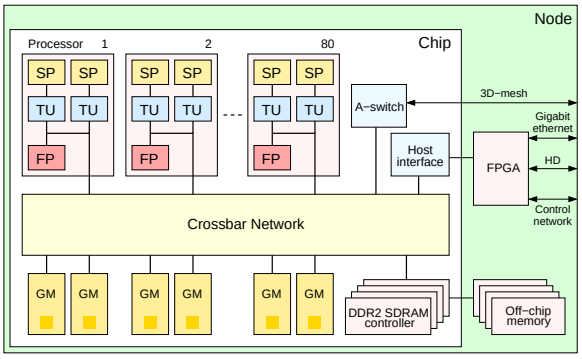

Figure 2. Cyclops-64 node

In Figure 2, we show the architecture of a C64 node. Each processing node consists of a C64 chip, external DRAM, and a small amount of external interface logic. Each C64 chip has 80 processors, each containing two thread units, a floating-point unit and two SRAM memory banks of 32KB each. A 32KB instruction cache, not shown in the figure, is shared among five processors. A thread unit is a simple 64-bit in-order RISC processor core with a small instruction set architecture (60 instruction groups) operating at 500MHz. In the C64 chip architecture there is no data cache. Instead a portion of each SRAM bank can be configured as scratchpad memory. Such a memory provides a fast temporary storage to exploit locality under software control. Processors are connected to a crossbar network that enables intra-chip communication, i.e. access to other processor's on-chip memory as well as off-chip DRAM, and inter-chip communication via the input and output ports

of the A-switch, which is the communication device that connects each C64 chip to its nearest neighbors in the 3D-mesh. The intra-chip network also facilitates access to special hardware devices such as the Gigabit Ethernet port and the serial ATA disk drive attached to each C64 node.

Finally, Figure 3 illustrates the physical arrangement of a C64 supercomputer, which corresponds to a $24 \times 24 \times 24$ logical configuration.

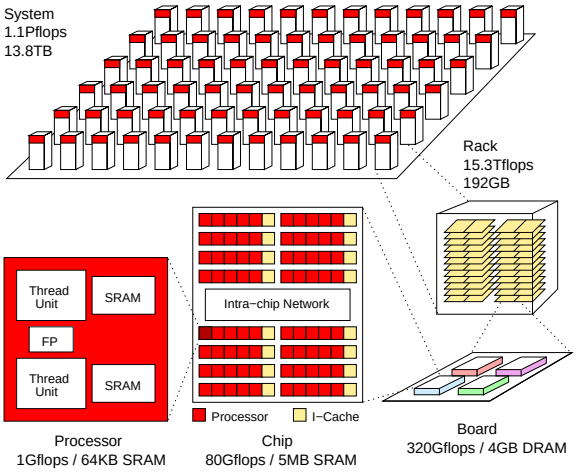

Figure 3. Cyclops-64 supercomputer

3. Cyclops-64 toolchain

Figure 4 illustrates the software toolchain currently available for application development on the C64 platform. The C/Fortran compilers have been ported from the GCC-3.2.3 suite, although a more recent porting from the GCC-4.0.2 is also available. Assembler, linker and other binary utilities are based on binutils-2.11.2. To fully exploit C64 multi-layered memory hierarchy, the toolchain is designed to support segmented memory spaces that are not contiguous. In other words, multiple sections of code, initialized and uninitialized data may be allocated on each memory region, just like in some toolchains for embedded processors. To direct the allocation of sections, pragmas are provided to specify the memory areas were the user would like to place certain variables or procedures. For instance, frequently used data structures can be put in the scratchpad memories, closer to the processor/thread units. In general, applications should be designed having in mind the on-chip and off-chip memories latency and bandwidth, such that in the end they make the best use out of the memory. The current toolchain with pragma support for segmented memory spaces is the first step towards this goal.

The C standard and math libraries are derived from those in newlib-1.10.0. Functions (libc/libm) are thread safe, i.e.

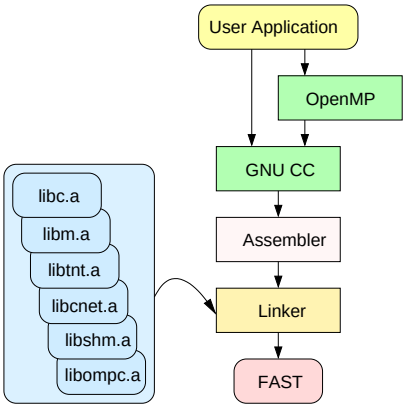

Figure 4. Cyclops-64 software toolchain

multiple threads can call any of the functions at the same time. Nonetheless, mutual exclusion is guaranteed by efficient spin locks. In addition, memory functions have been optimized taking into account the memory hierarchy and C64 ISA support for multiple load and store operations that make a more efficient use of the memory bandwidth.

The TNT microkernel/runtime system library provides the software and application developer with the functionality to write multithreaded programs: thread management, support for mutual exclusion, synchronization among threads, etc. In order to achieve high performance and scalability, the implementation of such functionality tries to match as close as possible the architecture underneath the microkernel/RTS, as explained in the next section.

The Cnet communication protocol is also part of the microkernel. This software component controls the A-switch, and supports SHMEM, a one-sided communication library, on top of it. SHMEM provides a shared global address space, data movement operations between locations in that address space, and synchronization primitives that greatly simplify programming for a multi-chip system such as C64.

To carry out our research until a hardware platform is available, we wrote FAST: an execution-driven, binary-compatible simulator of a multichip multithreaded C64 system. FAST accurately reproduces the functional behavior and count of hardware components such as chips, thread units, on-chip and off-chip memories, and the 3D-mesh network. Although FAST is not cycle accurate, we have shown it is useful for performance estimation [3].

4. Cyclops-64 thread virtual machine

As we described in the introduction, one important role of the C64 system software is to implement a virtual machine. This virtual machine, called C64 Thread Virtual Machine, can be viewed as a multichip multiprocessor extension to the base C64 instruction set architec-

ture. In this section, we present a refinement of the C64 Thread Virtual Machine along with its implementation, TiNy Threads^{TM}(TNT) [4]. TNT has been designed and developed to support a multithreaded programming model for a cellular multithreaded architecture such as Cyclops. Given C64 special features, it was not our intention to develop an OS for this platform that would put a considerable overhead on top of a machine that is aimed for simplicity from the bottom up. Instead we decided to implement TNT directly on top of the hardware architecture as a micro-kernel/run-time system library that takes advantage of C64 hardware features while providing an interface that shields application programmers and system software developers from the complexities of the architecture wherever possible. The C64 Thread Virtual Machine includes three components: a thread model, a memory model and a synchronization model.

4.1. Thread model

Although C64 architecture supports user and supervisor operation modes, execution is non-preemptive. That means once an application starts running on the C64 there is no mechanism available to interrupt the program unless an exception occurs. However, the C64 instruction set architecture design includes efficient support for thread level execution. For instance, it provides a sleep instruction, such that a thread can stop executing instructions for a number of cycles or indefinitely. While asleep a thread may be woken up by another thread through a hardware wake-up signal. Such a signal is generated when a store to a thread-specific memory-mapped port is executed.

In the TNT thread model, thread execution is non-preemptive and software threads map directly to hardware thread units. In other words, after a software thread is assigned to a hardware thread unit, it will run on that hardware thread unit until completion. Furthermore, a sleeping thread will not be swapped out so that idle hardware resources can be assigned to another software thread. As in other thread models, a waiting thread (waiting on an external event/synchronization) goes to sleep; such a thread is woken up by another thread through the hardware signal.

4.2. Memory model

On C64 there is no hardware virtual memory manager, which means the three-level memory hierarchy of the C64 chip is exposed to the programmer. The C64 hardware chip supports direct memory access from all thread units/processors to the shared address space covering the on-chip memory (interleaved and scratchpad sections) and the off-chip DRAM banks associated with the chip. That is, all threads see a single non-uniform shared address space. On-chip SRAM memory space is limited in the current technology to 5MB, so it should be viewed and used as temporary storage during computation. There is no hardware data cache used in the C64 design. Off-chip DRAM should be considered as the main memory. In addition, it has been proven that the C64 architecture behaves as sequentially consistent for the interleaved and off-chip memories. However, hardware cannot guarantee a "Lamport order" of the accesses to the scratchpad memory space, hence no sequential consistency can be assumed.

TNT is a memory-aware runtime library that takes advantage of C64 explicit memory hierarchy by placing frequently used data in scratchpad memories that are closer to the processor/thread units. Upon initialization each software thread is given control over a well determined region of the scratchpad memory, which is allocated to every physical thread unit at boot time. Such a section of memory holds the thread descriptor, a fixed-size structure (192 bytes) that holds all the information required to properly handle the thread, including its stack pointer, and a small amount of thread local data directly managed by the user.

4.3. Synchronization model

C64 architecture has a rich set of hardware supported in-memory atomic instructions. Unlike similar instructions on common off-the-shelf microprocessors, atomic instructions in the C64 only block the memory bank where they operate upon while remaining banks continue servicing other requests. In addition, threads within a C64 chip are connected to a 16-bit signal bus that provides a means for very fast communication of a small amount of information. Thread units also have an interthread interrupt device, which is mapped to memory. This device allows one thread to interrupt another (or itself).

TNT provides a unique spin lock algorithm for shared memory synchronization designed to make best use of the C64 architecture supported in-memory atomic instructions and thread sleep/wake-up mechanisms. Moreover, this spin lock implementation has been proven to outperform lock-free data structures on C64 [2]. For collective synchronization, TNT library provides direct access to the signal bus interface register. Besides significant improvements in the execution time of barrier operations, the signal bus reduces memory traffic and power consumption, as spinning waiting for a signal bus line to drop does not interfere with other thread units or generate excessive heat. A third type of synchronization in TNT is introduced to express precedence relations between operations from two different threads. In the first version of the C64 TVM, we provide a coarse-grain signal-wait type of synchronization based on the interthread interrupt that should be placed between a pair of specific program points within the two threads.

5. Host control software

This section outlines the system software that is specific to the host. We refer to it as the host control software and its three main components are: job scheduling and launching, resource manager and host to C64 communication.

5.1. Job scheduler

Like in other large computing systems [1, 6, 5, 7], the goal of the job scheduler is to maximize the utilization of the computing system by minimizing waiting and idle times. On C64, job scheduling supports both interactive and batch modes. A C64 system may be partitioned into development and production sections. For a fast turn-around, the development partition may be used interactively while the production partition is restricted to batch submission. In interactive mode users are granted access to a small number of C64 nodes for the purpose of debugging or tuning applications. Batch jobs submitted by users are put into a job queue by the queue manager process. Associated with each job there are certain parameters, including priority and resource requirements such as number of C64 nodes. Every time nodes in the production partition are released, the job scheduler is awaken and decides which job runs next. The decision is made based on the list of parameters submitted with the job as well as runtime factors such as time awaiting on the queue. In addition, the scheduler invokes a placement algorithm that determines the set of C64 nodes assigned to run a job. Placement accounts for faulty nodes and guarantees the number of nodes requested by the user.

5.2. Resource manager

A resource manager is deployed to manage the system resources, including C64 and front-end nodes. Its objective is to minimize downtime of the system due to hardware failures and hence, to improve the utilization of the system. On a system as complex as the C64 computing environment, the sources of failures are numerous. For instance, a thread unit, floating point unit or memory bank of a chip may be bad. An entire chip may be inaccessible due to a malfunction of the A-switch or some link of the 3D-mesh may not work as expected. The C64 system software in general and the resource manager in particular detect and try to work around all these and many other issues. For instance, at boot time each C64 node is thoroughly tested to determine its aptitude to run programs. C64 architecture provides a hardware mapping table (accessible to the resource manager only) where bad components are marked and effectively removed from the set of active elements. The resulting chip with a reduced number of resources is still eligible for computation. Similarly, faulty nodes and links may be assigned to partitions allocated to run jobs. However, these may be avoided by means of a routing algorithm. Given the nodes and links status information generated by diagnostics programs run under the resource manager control, the role of the routing algorithm is to find at least one path between any two C64 nodes in a partition and among C64 and front-end nodes. The ability to remain operational despite of hardware failures is unique of the Cyclops architecture and provides a cost-effective solution with unparallel efficiency among common off-the-shelf microprocessors-based supercomputing systems.

Additionally, the resource manager maintains a central database, which provides a reliable and comprehensive view of the system. Such information eases the design of the system software. For instance, the job scheduler requires the knowledge of bad chips to ensure the user requirements in terms of minimum number of working nodes are met. In the event of a hardware failure, for instance a C64 chip stops responding during the execution of a program, this view of the system allows recovery in minimum time. As soon as a node within the partition where the job was running is identified as faulty, the remaining C64 nodes are moved again to the pool of available resources. Notice that while the status of a partition is verified, other jobs may be assigned to other partitions independently.

5.3. Host to Cyclops-64 communication

The C64 supercomputer is attached to the host system through a number of Gigabit Ethernet links. These links, in addition to the 3D-mesh support all the communication between C64 and front-end nodes. Therefore, system software developers are required to handle the specifics of both Ethernet and Cnet protocols to carry out any communication successfully. To avoid this trouble an uniform communication protocol layer, called CDP, is added. CDP provides a global address space across the front-end host and the C64 back-end. Based on CDP, application level protocols are implemented, including file I/O, debugging, performance monitoring and host to C64 remote memory communication. For instance, when a C64 node attempts to open a file, a request is shipped to the front-end in the form of a CDP packet. At the host, a daemon performs the operation on behalf of the back-end and sends the result (file handler) back to the C64 node where the I/O operation came from. For file I/O the C64 computing engine always starts the transaction. However, there are services that are initiated by the front-end instead. For instance, when a job is scheduled to start execution on a set of C64 nodes, the job scheduler contacts the process control thread running on each C64 node and transfers among other information the program's image, the user environment, command line parameters, etc. All this data communication relies on the CDP protocol as well.

Table 1. Microprocessor parameters

Processor	Clock	Cache	Memory
Intel Centrino	1.86GHz	2MB L2 cache	512MB
AMD Opteron	2.4GHz	1MB L2 cache	3GB
Intel Pentium4	3.2GHz	512KB L2 cache	1GB
Cyclops-64	500MHZ	No data cache 5MB on-chip memory	1GB

When file I/O processing is expected to be intensive, it would not be judicious to allow the C64 side drive the computation. That would result in numerous I/O requests being shipped to the front-end that could easily make of the Ethernet links the bottleneck of the entire system. To cope with this situation, a different computing paradigm is supported, in which an application consists of two processes: one running on the front-end, another on the C64 back-end. The former is responsible for I/O and takes care of preprocessing and off-loading computation to the latter, which accomplishes the computational intensive part. Once computation is done, if any post-processing is required the front-end will handle it. We enable this scenario with a remote memory operations library (RMO) that facilitates inter-process communication. According to our current model, the application part running on the front-end cluster sends data to (push) and gets results from (pull) the C64-side. All the communication and synchronization primitives provided by the RMO library are implemented on top of CDP.

6. Experience

We report our experience with two case studies that demonstrate the cost-effectiveness of the C64 architecture compared to common off-the-shelf microprocessors.

6.1. Monte Carlo simulation

We start with a Monte Carlo simulation, which is a popular method for determining option prices. This finance application is computational intensive. However, it is also embarrassingly parallel, i.e. there would be little communication among entities (processes or threads) that participate in the parallel execution. We use this application to compare the cost performance of a C64 chip with other three microprocessors: Intel Centrino, AMD Opteron model 250, and Intel Pentium 4, see Table 1.

On all the platforms we run the sequential algorithm only. We use typical values as input parameters and take note of the execution times, see Figure 5. Since the application is embarrassingly parallel, a parallel implementation of the algorithm would yield linear speedup. Furthermore, the working data set is small enough to fit into a C64 thread unit's scratchpad memory. Hence, for the C64 chip, we estimate the parallel execution time as the execution time obtained from running the sequential algorithm on a thread unit divided by the number of thread units available on a C64 chip, 160 in total. Figure 6 represents the normalized execution time, which demonstrates that for this application a C64 chip delivers the same performance as 35 Intel Centrino, 25 AMD Opteron or 30 Intel Pentium 4, approximately. A C64 chip is estimated to cost the same as any of these other microprocessors. Additionally, a C64 chip with bad thread units due to some fabrication defect still can be used, increasing the yield and hence the cost performance of the C64 architecture.

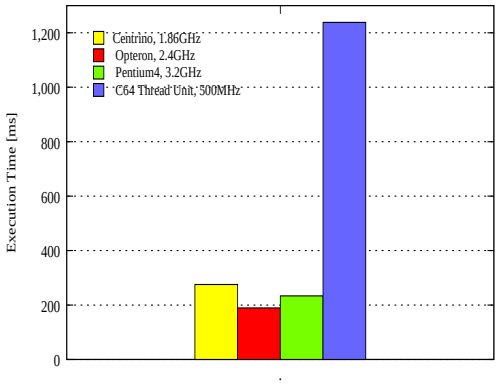

Figure 5. Monte Carlo simulation time

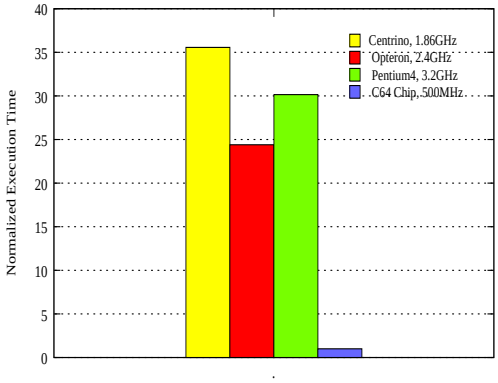

Figure 6. Normalized simulation time

6.2. FFT algorithm

The Fast Fourier Transform (FFT) is a well known algorithm widely used in a variety of areas, such as signal processing. On C64 we implement a one-dimensional double-precision complex transform whose data set fits into inter-

leaved memory. In Figure 7 we plot the Gflops obtained for FFTs with different number of data points. The performance approximates to 0.3 floating point operations per clock cycle per thread unit. For a small number of data points such as 2^{10}, performance flattens out after 16 threads, however for a 2^{16}-point FFT it scales up to 150 threads, at which point it delivers almost 20 Gflops.

We looked at the results published in http://www.fftw.org for other FFT implementations. We found that no conventional microprocessor can achieve a performance in excess of 4 Gflops, which is one-fifth of what a C64 chip delivers for the same cost. Although we followed the benchmark methodology described at the web site, the reader should be aware that routines that use different formats are not strictly comparable. Additionally, our results are only a preliminary estimation. Hence, this exercise should be regarded as a rough comparison of the relative merits of the C64.

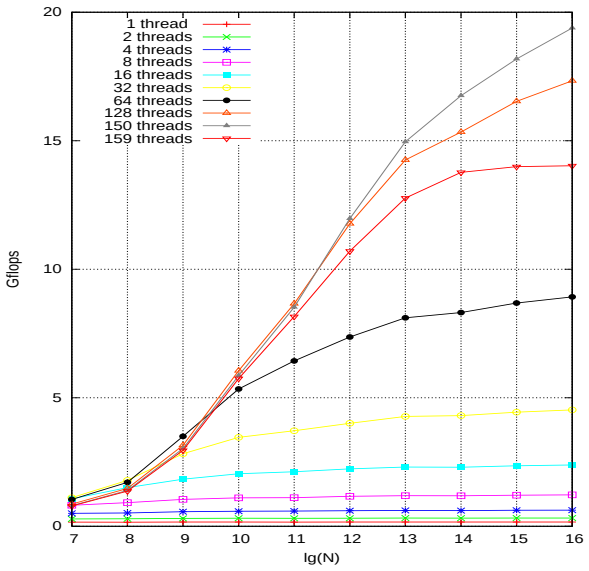

Figure 7. Gflops for the FFT on a C64 chip

7. Summary

In this paper we presented the first version of the C64 system software. First, we described the C64 software toolchain available for application development. As part of the toolchain we focused on the C64 Thread Virtual Machine, including its key components: the thread model, the memory model and the synchronization model, as well as its implementation: TiNy Threads^{TM}. We also outlined the management system software that runs on the host whose goal is to maximize the utilization of the C64 computing system. Finally, through two case studies we demonstrate how C64 provides a cost-effective solution for running computing intensive applications. For FFT and Monte Carlo simulation, a C64 chip's performance varies from 5 to 35 times faster than common off-the-self microprocessors.

8. Acknowledgments

We acknowledge the support from IBM, ETI, the Department of Defense, the Department of Energy (DE-FC02-01ER25503), the National Science Foundation (CNS-0509332), and other government sponsors. The authors would like to acknowledge all the people at the CAPSL group and ETI who have been involved in the Cyclops project, in particular Brice Dobry, Geoffrey Gerfin, John Tully and Wesley Toland. Special thanks to Parimala Thulasiraman and Ruppa K. Thulasiram from the University of Manitoba for providing the Monte Carlo code and Michael Merrill for the FFT implementation.

References

[1] R. Brightwell and L. A. Fisk. Scalable parallel application launch on Cplant. In *Proceedings of SC2001: High Performance Networking and Computing*, page 263, Denver, Colorado, November 10–16, 2001.

[2] J. del Cuvillo, W. Zhu, and G. R. Gao. Landing OpenMP on Cyclops-64: An efficient mapping of OpenMP to a many-core system-on-a-chip. In *Proceedings of the ACM International Conference on Computing Frontiers*, Ischia, Italy, May 2–5, 2006.

[3] J. del Cuvillo, W. Zhu, Z. Hu, and G. R. Gao. FAST: A functionally accurate simulation toolset for the Cyclops64 cellular architecture. In *Proceedings of the Workshop on Modeling, Benchmarking and Simulation*, pages 11–20, Madison, Wisconsin, June 4, 2005. Held in conjunction with the 32nd Annual International Symposium on Computer Architecture.

[4] J. del Cuvillo, W. Zhu, Z. Hu, and G. R. Gao. TiNy Threads: A thread virtual machine for the Cyclops64 cellular architecture. In *Proceedings of the Fifth Workshop on Massively Parallel Processing*, page 265, Denver, Colorado, April 8, 2005. Held in conjunction with the 19th International Parallel and Distributed Processing Symposium.

[5] S. M. Kelly and R. Brightwell. Software architecture of the light weight kernel, Catamount. Technical report, Sandia National Laboratories, Albuquerque, New Mexico, 2005.

[6] A. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for BlueGene/L systems. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, page 64, April 2004.

[7] J. Stearley. Towards a specification for measuring Red Storm reliability, availability, and serviceability (RAS). Technical report, Sandia National Laboratories, Albuquerque, New Mexico, May 2005.