BG/L Compute Node Kernel

Mike Mundy
IBM Rochester

February 2005 | Blue Gene/L

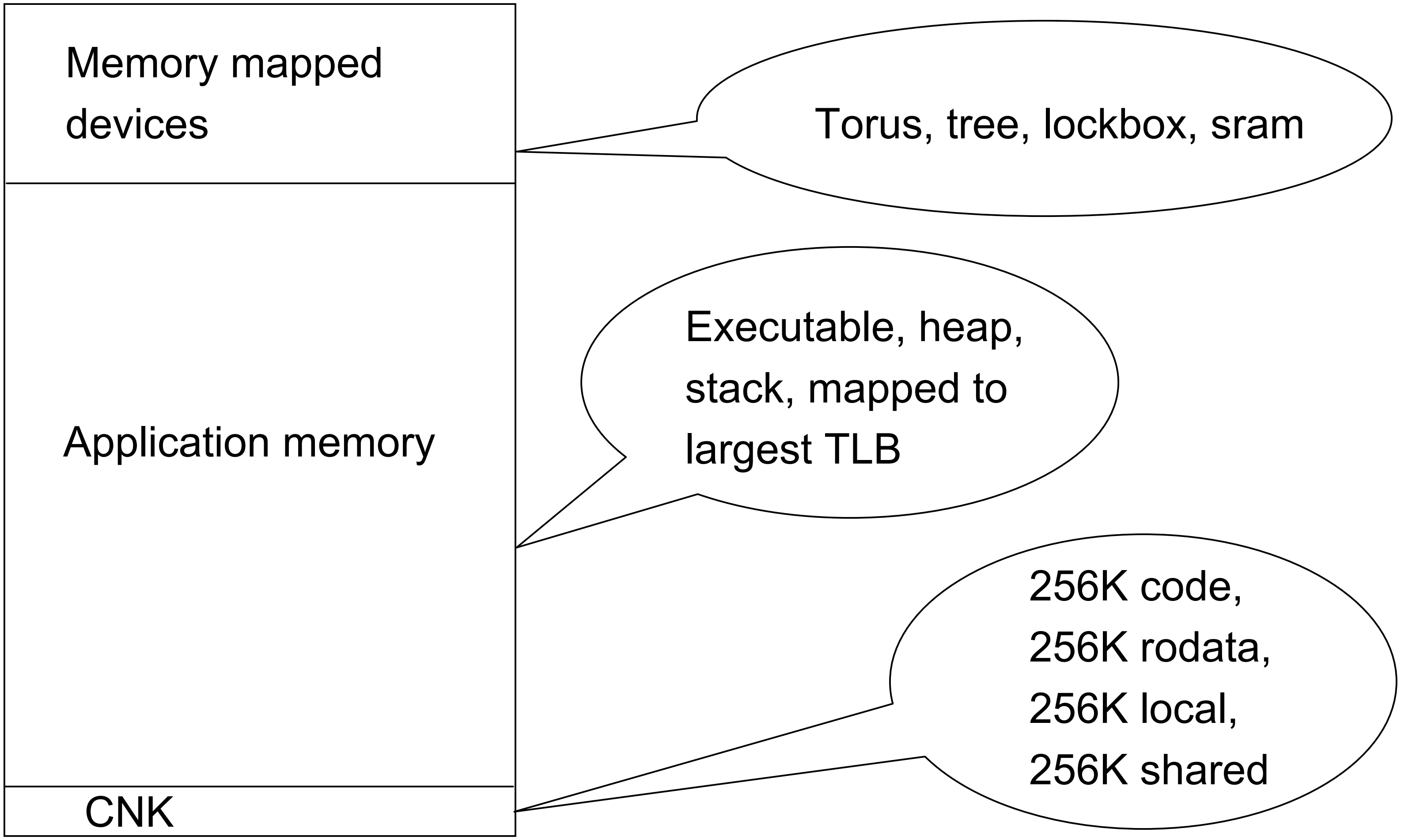
BG/L Compute Node Kernel Agenda

- CNK features and high-level design
- Function shipping to I/O node
- Booting compute nodes and managing jobs

CNK Features

- A simple Linux-like kernel
  - Runs one process at a time
  - Uses small amount of memory – rest for the application
  - Supports attaching debuggers
- CNK provides a subset of the Linux system calls
  - File I/O
  - Directory operations
  - Signals (ANSI C only)
  - Process information
  - Time
  - Sockets
- Goal is to stay out of the way and let the application run

Compute Node Memory Map



Memory mapped devices

Application memory

CNK

Torus, tree, lockbox, sram

Executable, heap, stack, mapped to largest TLB

256K code,
256K rodata,
256K local,
256K shared

CNK Modes

- Coprocessor mode
  - Application runs on processor 0
  - Very limited environment for running code on processor 1
  - MPI uses coprocessor for offloading communications
- Virtual node mode
  - Application is loaded and runs on both processors
  - Memory is divided in half
  - Application is responsible for sharing resources
- Mode is selected at boot time

CNK Limitations

- No support for asynchronous signals using sigaction()
- No support for Linux interprocess communication
- No support for server-side sockets APIs
- No support for poll() or select()
- Limited support for timers

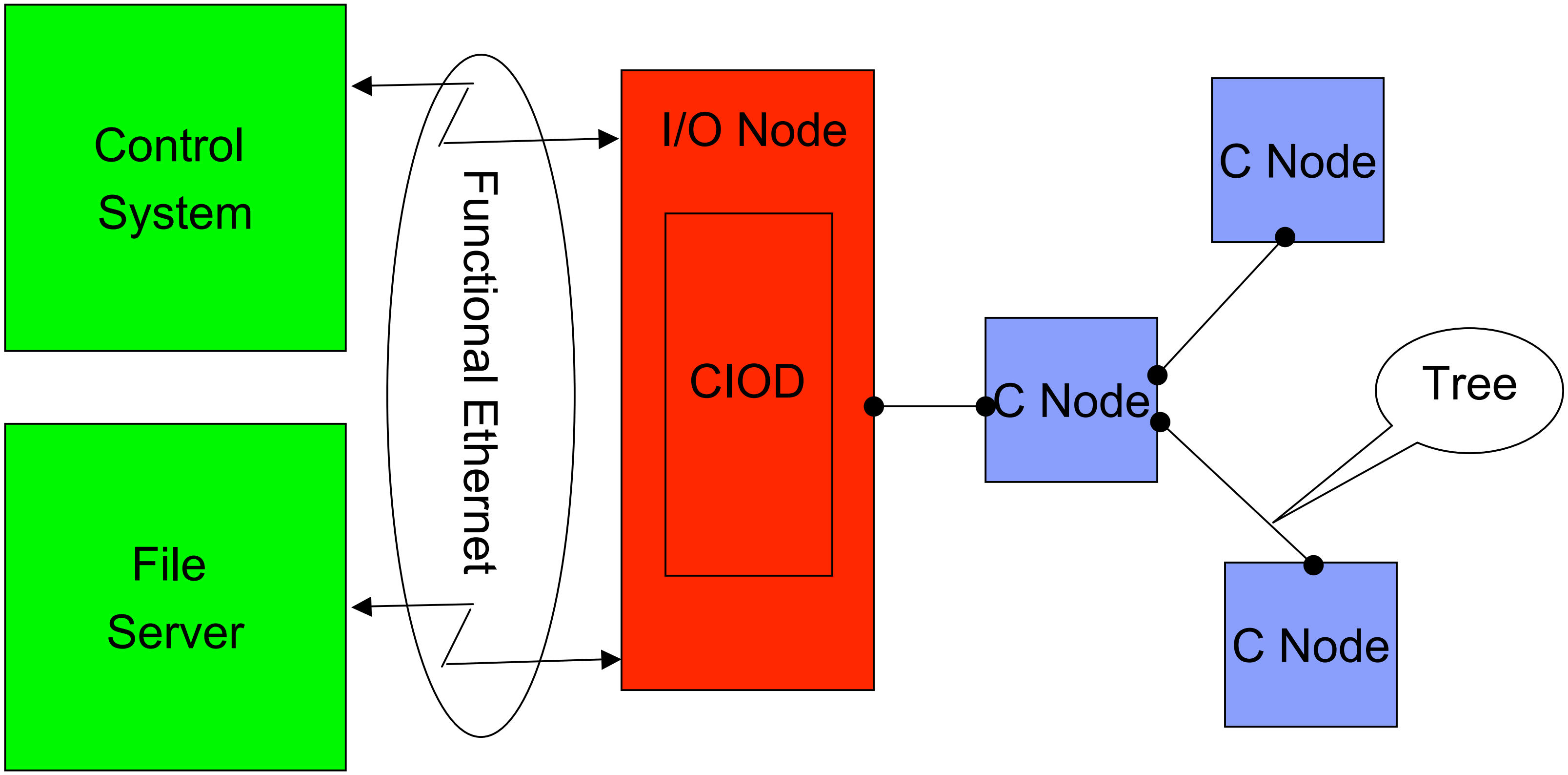
CNK Function Shipping

- All I/O must be processed on the I/O node
- CIOD is a user application running on the I/O node that:
  - Manages the compute nodes for the control system
  - Manages descriptors, working directory, umask for compute nodes
  - Performs all I/O for all compute nodes
  - Manages the debugger connections to the compute nodes
- Ratio of compute nodes to I/O nodes differs between machines
- All communication between CIOD and compute nodes is over virtual channel 0 of the tree network

CNK Function Shipping Example

- Application calls write() system call
- CNK breaks request into multiple messages
  - Size is configurable
  - Sends each message in turn to CIOD
- CIOD receives the message and calls write()
- CIOD sends result back to the compute node
- CNK collects the results from each message
- CNK returns result to application after either all of the data is sent or an error occurs
- CIOD never blocks on a system call
  - All sockets are implicitly non-blocking

CNK Function Shipping



Boot Process

- Control system starts microloader on compute nodes and I/O nodes
- Microloader boots CNK on compute nodes and Linux on I/O nodes
- Linux mounts file servers and starts CIOD
  - Customer can provide their own rc scripts
- CNK starts, initializes the compute nodes, sends message to CIOD
- CIOD starts, waits for all compute nodes to report, then waits for control system to connect

Job Startup

- CIOD accepts connection from control system
- Control system sends user login info and application info
- CIOD swaps to user
- CIOD reads application and sends to each compute node
- CNK loads application into memory and waits to start
- Control system sends start info
  - Debugger is optionally connected at start
- CIOD tells each compute node to start the application

Job Running and Termination

- CIOD forwards stdout and stderr text to control system
  - No support for reading from stdin
- Each compute node reports result to CIOD
  - Ended normally with exit status
  - Ended by signal with signal number
- CIOD forwards result to control system
- Control system waits for all compute nodes to end
- Control system closes connection to CIOD
- CIOD resets and waits for next job
- Control system can send signal to compute nodes
  - CIOD forwards to compute nodes

When things go wrong on I/O node

- CIOD is instrumented with trace points and status reporting
- If configured, CIOD listens on a service connection and supports commands to:
  - Turn tracing on and off
  - Report current status of compute nodes both summary and detailed
  - Report info about the tree network
- CIOD logs RAS events for error conditions
- CIOD tries to stay up and running even if an error occurs

CIOD Service Connection Example

```
telnet 172.30.60.152 7201
Trying 172.30.60.152...
Connected to 172.30.60.152.
Escape character is '^]'.
ciod running in coprocessor mode with 64 processors

> show_status
Mode: coprocessor
Job number: 1
Torus dimensions: X=8, Y=8, Z=8, T=1
Number of nodes: 64
Node   0:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   1:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   2:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   3:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   4:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   5:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   6:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
Node   7:  state=RUNNING      , ioState=NOT_WAITING       , debug wait=NOT WAITING
```

When things go wrong on compute nodes

- CNK logs RAS events for error conditions
- If application dies, CNK creates a text “core” file with:
  - Register contents
  - Call stack
  - Interrupt history
- CNK monitors tree and torus networks and reports status at job end

CNK Summary

- CNK is a simple Linux-like kernel
  - Subset of system calls
  - Two modes of operation
- CIOD manages compute nodes and performs file I/O
- Job startup and termination is driven by the control system