SAND2008-6151C

# Plan 9 on the BG/X systems

Ron Minnich
Sandia National Labs

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

# Overview

- Why
- Experiences with the port to newer systems
  - K8
  - BG/X
- Initial port feasibility testing
  - PPC 440
  - Mambo
- MPI Like
- CN/K compatibility environment
- Conclusions

Sandia National Laboratories

# Why?

- What Plan 9 is
- What some future HPC boxes look like
- Why it is a good match

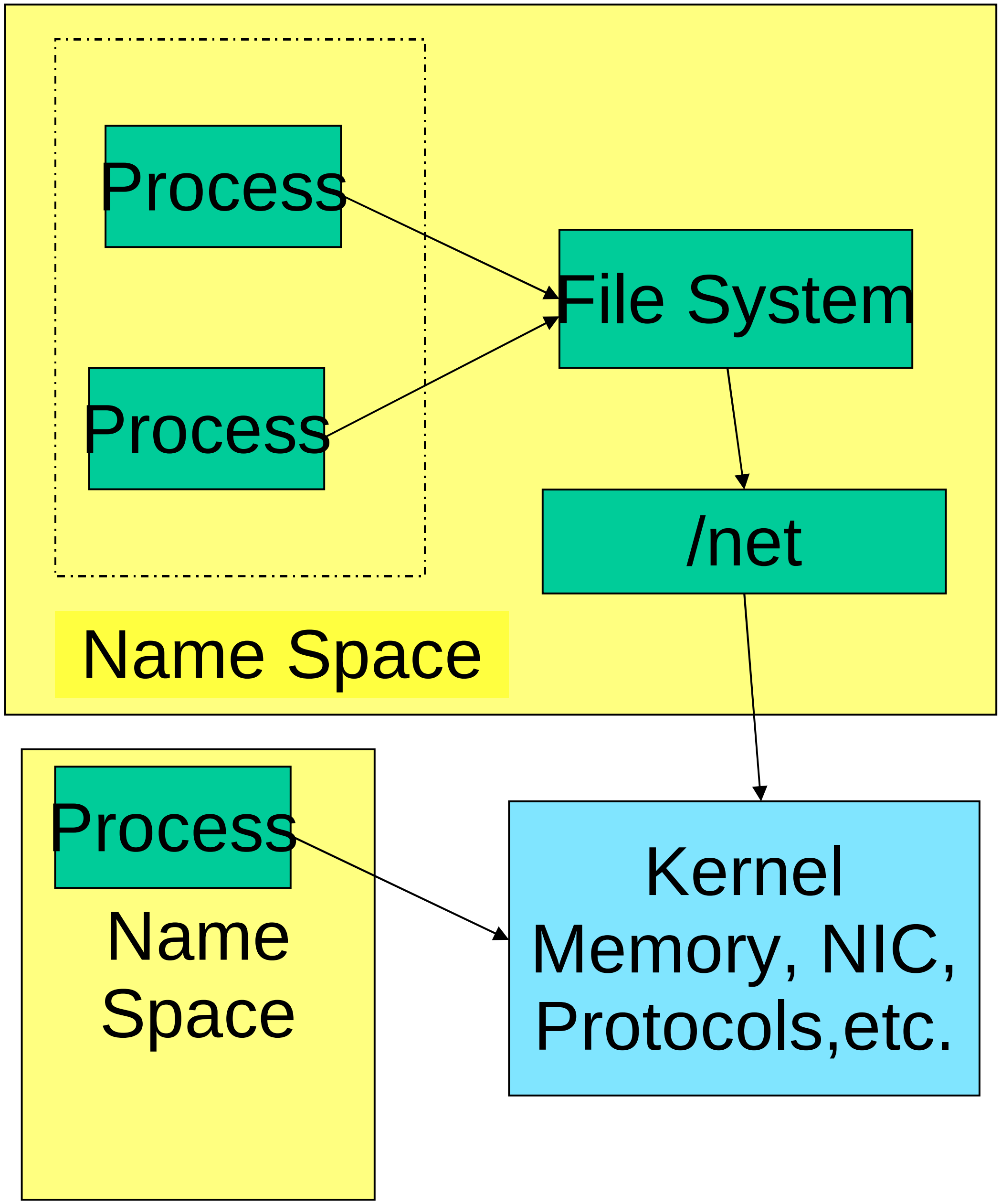
Sandia National Laboratories

# What Plan 9 is

- Not like anything you've seen
  - Not a mini-, micro-, nano-, or other fad kernel
- Core OS is fixed-configuration set of "devices"
  - Means "anything that *has* to be in the OS"
  - E.g. Memory,TCP/IP stack, Net hardware, etc.
- Everything else is a "Server"
  - File systems, windowing systems, etc.

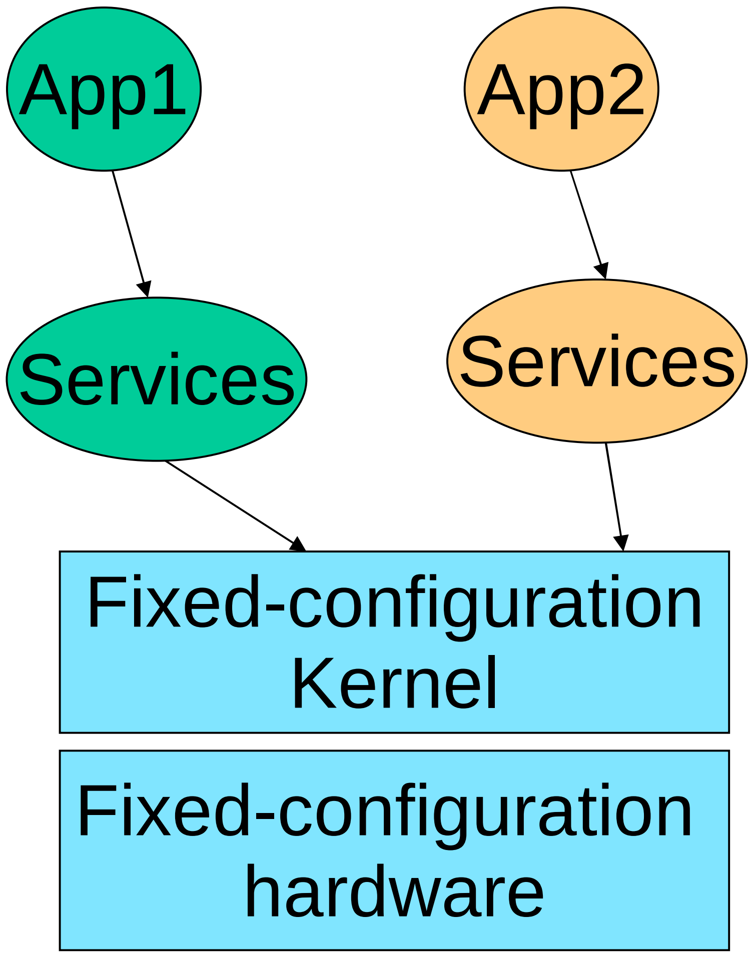
Sandia National Laboratories

# Plan 9 structure




- Processes attach *servers* as needed
- Attaches are inherited
- Not visible outside the group
- In this example one group has attached remote files
- Other group only needs IPC so it has no other services

# Why this is a good match to future HPC systems




- Future HPC machines feature fixed-configuration CPU/nodes
  - I.e. NO “hot plug”
- All variability is in software services used by apps
- Plan 9 fits this model perfectly
- Fixed kernel & hardware
- Customized services

# Advantages

- What has to be in the OS goes in the OS
- Everything else is optional
  - If you need something you pay for it
  - If not, not
- Options are configured per-process-group
- The name for this is “private name spaces”
- The name confuses people

# A way to think about private name spaces

- In the old days, all *memory* on a machine was shared *globally* by all apps
- That's how almost every OS extant does *files and servers* now
  - e.g. NFS mounts are visible to all
- Plan 9 provides a notion of private file system name spaces analogous to private memory space as introduced ca. 1955

Sandia National Laboratories

# File System Name Space types

**Global**

- All mounts visible to all processes
- On Unix, any proc can get to any file
- Mounts affect global state
- As if all programs shared all variables

**Private**

- Mounts visible in process group
- Only procs in the group can get to files
- Mounts affect group state
- Private variables

# And did I mention there are advantages?

- 38 system calls
- Linux is at ~~240~~ ~~280~~ 300 and counting
- Other non-Linux efforts have too-limited capabilities
- Plan 9 got modularity right

# What modularity is

- This is a John Deere tractor Power Take Off
- Connects to *modules*
- Modules stay the same for *decades*
- A very old module fits a very new tractor

# Software modularity

- Plan 9 kernel system call set:
- BIND CHDIR CLOSE DUP ALARM EXEC
- EXITS FAUTH SEGBRK OPEN OSEEK SLEEP
- RFORK PIPE CREATE FD2PATH BRK_ REMOVE
- NOTIFY NOTED SEGATTACH SEGDETACH SEGFREE
- SEGFLUSH RENDEZVOUS UNMOUNT SEMACQUIRE
- SEMRELEASE SEEK FVERSION ERRSTR STAT FSTAT
- WSTAT FWSTAT MOUNT AWAIT PREAD PWRITE

Sandia National Laboratories

# Plan 9 modularity

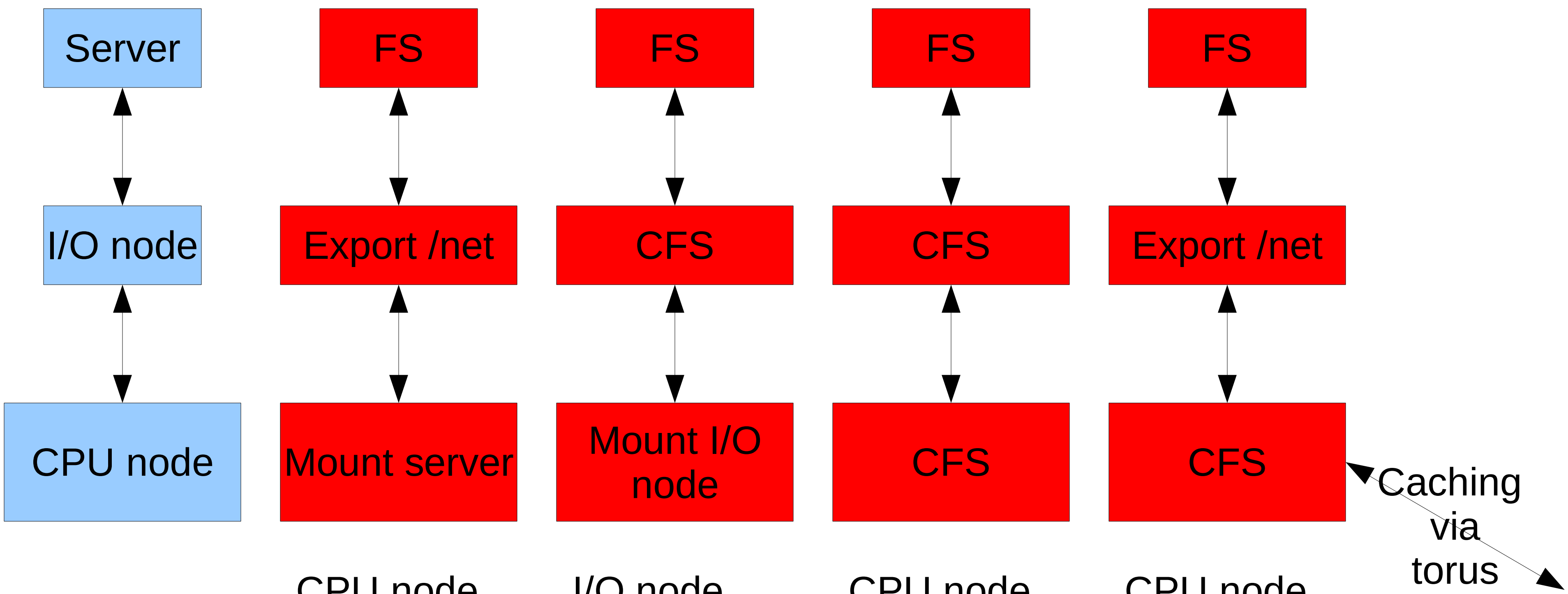
- Any server that uses that system call set works on any version of Plan 9
- It has worked this way for 18 years
- Typically only 6 calls are used: open close read write mount bind
- Servers are location-independent
  - So you can move them around as needed
- Which means that we can balance bandwidth, sharing, and latency when locating a server

# Balancing act

- You might want a more central server to optimize caching
- You might want to locate server components in the HPC fabric for latency and bandwidth
- It is trivial in Plan 9 to layer servers to achieve these effects

# Modularity example: how to access files

These four scenarios show different ways of connecting file servers to CPU node processes. None require special privileges.




CPU node imports /net from I/O node, mounts server

I/O node mounts server cache FS, CPU node mounts I/O node

CPU node starts cache FS, mounts I/O node

CPU node imports /net from I/O node, starts cache FS, mounts server

Sandia National Laboratories

# Experiences with the port

- In June 2006, we had been working with Plan 9 for about a year
- The experiences were interesting

# Plan 9 port to K8 – 2 phases First 64-bit port

- basic port
  - (running a shell, connecting to network, etc.)
  - took about 2 months (not full time).
  - mostly done in parallel with compiler.
- first phase resulted in what was essentially an x86 with 'fat' pointers -
  - vm layout and restrictions the same as an x86.
  - this let us become familiar with the compiler and hardware without fighting broken utilities.

Sandia National Laboratories

# Phase 2

- second phase was fix the programmes identified by compiler
  - warnings ("conversion of pointer to shorter integer")
  - fix the kernel system call linkage to deal with arguments which are a mixture of 32 and 64 bits.
- 164 files excluding kernel had compiler warnings.
- most were easy to fix by declaring the type of a variable correctly.
- Added type safe linker for kernel and user
  - Which showed that Python is not type-safe

Sandia National Laboratories

# Other stuff

- one just had to be hacked horribly, lex,
  - for which the man page already said 'The asteroid to kill this dinosaur is still in orbit'.
- some showed abuse of interfaces, e.g.
  - if(p = (Proc*)setjmp(_mainjmp))
  - and some showed failure of vision in the specification of some of the more esoteric plan9 system calls, e.g. rendezvous.

Sandia National Laboratories

# More other stuff

- Symbol tables and exec headers had to become 'fat'
- Compilers/debuggers had to understand.
- Mostly in a single library
  - and the kernel 'exec' system call.
- But it all worked ...

Sandia National Laboratories

# In June 2006 we got the word

- "Drop that cluster work"
- "We have bigger problems, i.e. a big BG/P coming along"
- "We need solutions that are not
  - Another Light Weight Kernel
  - Another Linux"
- So we changed direction
- Discussion with IBM revealed that BG/L was a good target (and there was interest)

Sandia National Laboratories

# We Started in August 2006

- Started with Inferno 405 port
  - Plan 9 derived OS for small embedded systems
  - Has no user mode, limited MMU use, hence easier
- 1 week in August: port to PPC 440
  - MMU, drivers, etc.
- Then a week to boot on BG/L CPU
- Then a week to do networks
- Then polishing up via email and IRC
- 4 people x 4 weeks (really!)

# Total port effort for June 2007 demo

- 16 man weeks
- How much assembly in Plan 9 kernel?
  - 1033 lines
- How many files in Plan 9 BG/L kernel?
  - About 90, including auto-generated by config
- 18 are platform-specific
  - Of which we had to modify about 10
- I realize that “file count” is somewhat bogus, but interesting

# Development

- All development is cross development
- A few key decisions make it easy
  - Here's a simple one: object file types for different architectures have a different suffix
- No complex path and environment mangling
- On a reasonable K8, kernel builds in a few seconds
- Next step is to build kernel on BG/L

# How current BG/L is set up

- Two kinds of nodes in BG/L: Linux IO nodes, CNK CPU nodes:
- e.g. LLNL: 1024 IO nodes, 64 CPU nodes per IO node, 2 CPUs per node, 128K+2K in all
- BG/L networks are several:
  - Ethernet to I/O nodes,
  - Tree to all nodes
  - Torus on CPU nodes only

Sandia National Laboratories

# Current file IO

- IO nodes talk to file servers -- Ethernet
- CPU nodes talk to IO nodes – tree
- The tree is interesting
- Has 16 “Classes”
  - Essentially a broadcast medium like unto coax
- Class 0 is set up for CPU <-> IO
- Class 1 is for CPU <-> CPU

Sandia National Laboratories

# Interconnect - Light Weight Protocol & Interfaces

- Existing software gives two options
  - CNK – no interface, software accesses hardware directly
    - Well, sort of.  MPI runtime actually has a lengthy call path
  - Linux – full socket abstraction and TCP/IP stack with lots of extra fluff (why do you need to ARP when you know where everyone is? And why have full sliding window protocol when you have h/w reliability mechanisms & flow control)
- Existing choices are both heavy weight in their own way due to unnecessary complexity in the stack.
- Proposed Solution
  - Use tailored light weight protocols & interfaces which leverage underlying hardware properties

# What net interface for apps?

- BG/L idea is direct application access
- But: can't do multiple apps with direct
- Why direct? Assumed overhead of an OS
- Fall 2006, we measured time from app pwrite()->kernel->wire
- Use sim and native tools and got output that looks like this:

# Output

| | |
|---|---|
| acid: 0x0119dd39 | n = r;==>/9k/port/sysfile.c:790 |
| acid: 0x0119dd3a | n = r;==>k/port/sysfile.c:790 |
| acid: 0x0119dd3b | off = ~0LL;==>9k/port/sysfile.c:792 |
| acid: 0x0119dd3c | off = ~0LL;==>9k/port/sysfile.c:792 |

etc.

- About 600 ticks
- About 180 lines
- Comparable to overhead for an OpenMPI send

# So, given a low overhead OS

- The need for OS bypass is unclear
- Modeled all interfaces as Plan 9 network interfaces
- Note: NOT ethernet interfaces, as done in Linux
  - NETWORK interfaces
- In particular, Plan 9 NETWORK interfaces don't require ARP; Linux ETHERNET interfaces do
- No need for 6-octet MAC address as in Linux
- So we don't need 20,000 entry ARP table as on XT4 systems

Sandia National Laboratories

# Tree addressing on Plan 9

```
switch((th->ipv4src[0]<<8) | th->ipv4dst[0]){
	case (IOdot<<8) | CPUdot:
		hdr = MKTAG(IOtoCPU, 0, PIH_NONE);
		break;
		/* etc. */
```

- We can map directly from IP to network address (or, in this case, class)
- Torus case is similarly simple.
  - Direct IP/MAC mapping
- No ARP tables! No /etc/dhcpd.conf! No /etc/hosts! No per-node files of any kind!

Sandia National Laboratories

# Network IO

- The IP mode is a stopgap
- Next steps are to play some tricks
- Example: tag is 20 bits (or so)
- So, on CPU->IO send, use tag type packets and put CPU address in tag
- On IO->CPU send, use p2p type packets and put CPU address in p2p
- P2P does not save network BW, just interrupts

Sandia National Laboratories

# File system IO

- On BG/L, CPU does IO via system call forwarding
- Not needed on Plan 9
  - Just import file system from IO node
- General mechanism replaces a complex, specialized one
- And it “just works”, from day one (it's almost boring)

Sandia National Laboratories

# Compiler

- Vita Nuova has added FP support that automatically operates the HMMR 2 chip

# Giant pages

- Exploit VM big pages: Right question
- Hugetlbfs: wrong answer. In fact, most Linux answers in this area are wrong
- VM subsystem should automatically align memory allocation, page alignment, from set of choices

# Not Huge Pages, Right Pages

- the plan 9 mmu code is ~1600 lines of machine independent code
  - ~400 lines of machine dependent code (independent of underlying hardware)
- will use superpage promotion rather than relocation.
- should be integral to the core of the o/s, not a bag on the side.

Sandia National Laboratories

# Right pages

- the machine independent code is ~16 years old, time and architectures change.
- Plan to completely rewrite bearing in mind
  - Modern architectural trends
  - Superpages
  - Large, sparse address spaces

# 6/07 Obligatory screen shot (10am)

```
EVH011_32_NE-0x00000fa5# cpu -h /net/tcp!11.0.0.1!17010 -a none
# mount /srv/io /n/io
# /n/io/power/bin/ps
bootes            1    0:00   0:00      204K Pread    boot
bootes            2    0:05   0:00        0K Wakeme   genrandom
bootes            3    0:00   0:00        0K Wakeme   alarm
bootes            5    0:00   0:00      268K Pread    paqfs
bootes           12    0:00   0:00        0K Wakeme   rxmitproc
bootes           21    0:00   0:00        0K Wakeme   torusread4
bootes           25    0:00   0:00        0K Wakeme   treeread
bootes           31    0:00   0:00        0K Wakeme   loopbackread
bootes           42    0:00   0:00      160K Await    listen
bootes           44    0:00   0:00      160K Open     listen
bootes           45    0:00   0:00        0K Wakeme   #I0tcpack
bootes           46    0:00   0:00      160K Open     listen
bootes           47    0:00   0:00      160K Open     listen
bootes           58    0:00   0:00      120K Await    tcp17010
bootes           59    0:00   0:00      264K Await    rc
bootes           60    0:00   0:00      120K Pread    tcp17010
bootes           65    0:00   0:00      196K Pread    ps
#
```

# Conclusions from 2007

- Plan 9 is up and working on BG/L
- It's low noise, but featureful
- Initial system uses IP for networks, but not via hoary “everything is an ethernet” approach
- System call overhead is low; do we need direct access?
- Apps testing starts now

# One year later ...

- Had done some initial port of simple apps to Plan 9
- Developed an “MPI Like” library
- Determined that we needed binary compatibility
- Started the BG/P port

# MPI usage for two apps

- MPI_Init 2
- MPI_Initialized 1
- MPI_Finalize 5
- MPI_Comm_rank 8
- MPI_Comm_size 8
- MPI_Comm_split 6
- MPI_Comm_dup 2
- MPI_Barrier 41
- MPI_Bcast 171
- MPI_Allreduce 39
- MPI_Send 24
- MPI_Recv 24
- MPI_IRecv 18
- MPI_ISend 18
- MPI_Waitall 15

- MPI_Type_struct 1
- MPI_Type_commit 16
- MPI_Type_vector 15
- MPI_Alltoall 6
- MPI_Gather 6
- MPI_Scatter 2
- MPI_Get_count
- MPI_Op_create
- MPI_Reduce
- MPI_Op_free
- MPI_Errhandler_set
- MPI_Wait
- MPI_Rsend
- MPI_Irsend

Sandia National Laboratories

# Examined usage and code

- First test was on HPCC apps
- Chose GUPS
  - Expected it to be a worst case
  - Assumed it would be simple code
  - Low “surface/volume”

Sandia National Laboratories

# MPI Like

- Simple library that can support several HPCC applications
- Relies on a few basic primitives
- And some Plan 9 library capabilities
  - Lock free threads
- And Function pointers (really!) and Sizeof (honest!)
- And gets rid of a lot of MPI wordiness
  - e.g. MPIDOUBLE etc. etc. (that's where sizeof comes in)

Sandia National Laboratories

# Basic data types

```
 struct Tpkt
{
     u8int sk;                /* Skip Checksum Control */
     u8int hint;              /* Hint|Dp|Pid0 */
     u8int size;              /* Size|Pid1|Dm|Dy|VC */
     u8int dst[N];                 /* Destination Coordinates */
     u8int   _6_[2];                    /* reserved */
     u8int session;
     u8int tag[4];
     u8int rank[2];
     u8int unused;
     u8int payload[];
};
```

- Not visible to programmers!

# Torus instance

```
struct TorIO
{
    int fd;
    int len;
    int myproc;
    int numprocs;
    struct Tpkt *map;
    struct Tpkt pkt;
};
```

- File descriptor for I/O
  - i.e. not mmap
- Map for other nodes
- One packet for reception

- Not direct access
- Len tells how many nodes (and map size)
- Only one receive struct for now
- Should probably make it an array

# Using the library

- struct TorIO *newTorIO(int fd, struct Tpkt *map, int len, int myproc, int numprocs)
  - Allocate a struct for torus IO

•Int sendtorus (int fd, Tpkt *pkt, void *data, int size);

•int recvtorus(int fd, Tpkt *pkt, int max);

- Send and receive data on the torus

•Int reduce (struct TorIO *tio, void *source, void *dest, int size,

void (*op) (void *, void *, int));

- Send to 0; 0 does the op; receive from 0

•Void intsum (void *dest, void *new, int);

- Apply sum to two int arrays

•Void dmax (void *dest, void *new, int);

- Apply max to two float arrays

Sandia National Laboratories

# Barrier with MPI Like

```
void
barrier (struct TorIO *tio)
{
    int dontcare;
    reduce (tio, &dontcare, &dontcare, sizeof (dontcare), nil);
}
```

# Basic GUPS loop

- Startup rank 0 with argv having list of nodes.
  - Start up other ranks with rank#, total ranks
- Rank 0:Send array of [x,y,z] coords
- Kick off threads:
  - One Recvthread receives updates. It gets the updates and increments the update count until it blocks or quits
  - Main thread works, sends updates as needed via non-blocking IO
  - Improvement: send thread per remote node

Sandia National Laboratories

# Why easier than MPI?

- (some) MPI programmers implement threads with counters and loops
  - Code is frequently hard to parse
  - e.g. GUPS was utterly unreadable
- Why manually encode sizeof()?
- Don't do XX_reduce – reduce can be polymorphic even in primitive language(C)
- The hardest part:reducing imcomprehensible code to lock-free-threads, simple structures

# MPI Like is a longer term project

- We do not want to imply that all code is as bad as GUPS code
  - But GUPS is not necessarily atypical
- But port effort is likely to be large
- And we lose XLF and XLC
- It is unlikely that we can bring programmers into this new environment absent non-zero effort
  - And even if it improves the code, they won't like it
- We need binary support

Sandia National Laboratories

# CNK emulation

- What would it take to run CNK binaries on Plan 9?
- It turns out not be as hard as might seem
- Issues:
  - Elf binaries
  - Only one syscall vector (as opposed to many on x86)
  - Different arg passing conventions
  - And, of course, the system calls

# What we did

- Elf converter (easy)
- Only one syscall vector: make variant proc type
  - Extend proc struct so we can mark processes as "cnk procs"
  - Proc can only mark itself to transition on exec
  - Transition once the process execs and not before
- Different arg passing conventions
  - Shim in syscall trap code
- And, of course, the system calls
  - Use Plan 9 syscalls where possible

Sandia National Laboratories

# Transition via exec

- We create a way to 'mark' a process as a cnk process
  - Add variables to arch-dependent part of proc struct
  - Add a control file to arch driver ('cnk')
- To make a process as 'cnk on exec'
  - Echo '1' > /dev/cnk
- In kernel:

```
up->cnkexec = 1;
```

# Starting up the cnk proc

```
up->cnk = up->cnkexec;
up->cnkexec = 0;
if (up->cnk) {
        ulong *l = &ureg->r7;
        int i;
        /* set up registers for CNK */
        ureg->r3 = nargs;
        ureg->r4 = (ulong) (sp + 1);
        ureg->r5 = ureg->r4; /*0; /* envp */
        ureg->r6 = 0;
        for(i = 7; i < 32; i++) /* poison */
                *l++ = 0xdeadbeef + (i*0x110);
}
```

- Copy cnkexec to cnk and clear cnkexec
- Linux expects nargs in r3 on startup
- Set envp
- Poison is very useful to catch bad behavior
- On return to user mode, syscall code paths change

Sandia National Laboratories

# System call switch on proc type

- Handled in trap()
  - cnk variable redirects system calls
  - We *could* just renumber the plan 9 system calls however
  - But there are other reasons to mark a process as 'cnk'

```
trap(int type){
switch(type) {
case INT_SYSCALL:
        if (up->cnk)
cnksyscall(ur);
        else
syscall(ur);
```

Sandia National Laboratories

# Other reasons to make a process as cnk

- May want to distinguish fault management handling
- Can have debug action depend on up->cnk
- Direct hardware access for programs
- We will probably add a tlb entry for cnk processes so they can address torus, tree, gib
- Another option is to wait until they fault, examine address, add proper tlb entry

Sandia National Laboratories

# cnk syscalltab

- Array of structs defining system calls
- Declare the syscall

```
Syscall cnkuname;

struct {
        char*   n;
        Syscall*f;
        int narg;
        Ar0     r;
} cnksystab[] = {
[122]   {"cnkuname", cnkuname, 1, {.i = -1}},
```

- Hence can index this table by syscall number for printname, func ptr, nargs, and default return value

Sandia National Laboratories

# Sample system call: cnkuname

```
Void cnkuname(Ar0*ar, va_list list)
{
    void *va;
    va = va_arg(list, void *);
    validaddr(PTR2UINT(va), 1, 0);
    memmove(va, "BGP\0plan9\02.6.19.2\0CNK\0 1\0", 26);
    ar->i = 0;
}
```

- Pattern: cast va_list to type; validate memory addresses; set return value
- For Plan 9 calls, it's easier: go direct to the call
  - e.g. pwrite()

Sandia National Laboratories

# Arg passing conventions

- syscall table is sparse

```
if(scallnr >= ncnksyscall || cnksystab[scallnr].f == nil){
	error(Ebadarg);
}
up->psstate = cnksystab[scallnr].n;
linuxargs[0] = ureg->r3;linuxargs[1] = ureg->r4; linuxargs[2] = ureg->r5;
linuxargs[3] = ureg->r6;linuxargs[4] = ureg->r7;linuxargs[5] = ureg->r8;
cnksystab[scallnr].f(&ar0, (va_list)linuxargs);
```

Sandia National Laboratories

# The big win

- CNK procs have direct access to Plan 9 syscalls
- Which means they can transparently use Plan 9 private name spaces
- Binary emulation provides us with a bridge to Plan 9 capabilities
- Less than 100 lines of changes to bgp-specific kernel code
- No CONFIG_CNK_EMULATION needed

# BG/P status

- Barrier is similar
  - Working now
- Tree is pretty much the same
  - Working now
- Torus is similar at bottom but has many new capabilities such as dma
- Ethernet is quite different
- Minor CPU differences

Sandia National Laboratories

# BG/P Approach

- Build a small “kernel” that is really a main with code to poke things
- Get console up first
- Start pushing various buttons with “kernel”
- In parallel with this work, start bringing BG/L kernel forward
- Also develop CNK emulation on PPC 440 board
- Had an initial boot in 5 days of work at Argonne

# Status

- Tree, barrier, working
- Torus dumping status info
- Ethernet still refusing to talk to us (X* interfaces are new territory)
- Binary emulation failing in getenv() (!)
  - After we resolved many other issues
- Working from public code so there are limits
- Hope to run mpihello by SC 08

# Conclusions

- We feel Plan 9 is a good match to future HPC
  - No USB or IDE ports on HPC nodes
  - Lots of flexibility in configuration
- Port to BG/L took lots of thinking but total work was not overwhelming
- Port to BG/P in progress
- Plan is to support binary emulation for CNK programs
- We can make Plan 9 power available to CNK procs

Sandia National Laboratories