K42 and Blue Gene
Two Case Studies for Parallel OS

Hermann Härtig
SS 2008
K42, Blue Gene, MosiX

K42
- Shared Memory SMP
- Emulate Linux interface
- Optimise locality and concurrency

Blue Gene
- Distributed Memory MPP
- Message Passing Interface
- Partition

MosiX
- “Clusters” with COTS networks
- Distribute Linux
- Balance Load dynamically
Overview

- Introduction and some terminology
- (Interconnect Architectures)
- Programming Models
- SMP operating systems case study: IBM K42
- MPP operating systems case study: IBM Blue Gene
- Cluster operating system case study: MosiX
- An SMP technique in detail: RCU by Frank Mehnert
SMP: Shared Memory / Symmetric MP

- Characteristics of SMP Systems:
  - Highly optimized interconnect networks
  - Shared memory (with several levels of caches)
  - Size today: up to ~1024 CPUs

- Successful Applications:
  - Large Linux (Windows) machines: (workstations and) servers
  - Transaction-management systems
  - Unix-Workstation + Servers

- Not usually used for:
  - CPU intensive computation, massively parallel Applications
MPP: Massively Parallel Multiprocessors

- Characteristics of MPP Systems:
  - Highly optimized interconnect networks
  - Distributed memory
  - Size today: up to few 100,000 CPUs

- Successful Applications:
  - CPU intensive computation, massively parallel Applications, small execution/communication ratios

- Not optimal for:
  - Transaction-management systems
  - Unix-Workstation + Servers
“Clusters”

- Characteristics of Cluster Systems:
  - Use COTS (common off the shelf) PCs and networks
  - Size: No principle limits (-> “GRID” computing)

- Successful Applications:
  - CPU intensive computation, massively parallel Applications, larger execution/communication ratios
  - Cooperation between large organisations

- Not optimal for:
  - Transaction-management systems
  - Unix-Workstation + Servers
LinPack Benchmark

- Jack Dongarra
- used for Top 500 list
  - largely superceded by LAPACK
- Selection of Fortran subroutines
  - analyse and solve linear equations
    - Matrixes: general, banded, symmetric indefinite, symmetric positive definite, triangular, tridiagonal square
  - column oriented access
  - analyse and solve least-square problems
    - QR and singular value decomposition
Top 500 List

Rank 1
Site: DOE /NNSA/LLNL (US)
Computer: Blue Gene/L IBM
Processors: 131072
Interconnects: Torus

- **Example: Cray T3D/T3E**
  - No shared memory
  - Supercomputer as front-end
  - High efficient network:
    Cache : Local : Distant
    3:27:220 Cycles
  - Embed distant memory in local address space in cache-line granularity.
  - scales up to 2K knots
Interconnects: Fat Tree

- pioneered in Thinking Machines CM5
Interconnects: Fat Tree

- pioneered in Thinking Machines CM5
Parallel Programming Models

- Organisation of Work
  - Independent, unstructured processes (normally executing different programs) independently on nodes (make and compilers, ...), “pile of work”
  - SPMD: single program on multiple data
    - asynchronous handling of partitioned data
      - (SIMD: same operation on different data, old MPPs)

- Communication
  - Shared Memory, shared file system
  - Message Passing:
    - Process cooperation through explicit message passing
Usage- and Programmingmodel

- SPMD

  ```java
  while (true) {
    work
    exchange data (barrier)
  }
  ```

- **Common for many MPP:**
  All participating CPUs: active / inactive

- **Techniques:**
  - Partitioning (HW)
  - Gang Scheduling
Distributed Shared Memory

- **Goal:**
  - Virtually Shared Memory

- **Problems:**
  - false sharing
  - Overhead

- **Solutions**
  - Weakened consistency model
  - Replication
  - Structured Memory Models (Tupel, Objects)

- so far: no success in practice !!!
Distribution of Load

- Static
  - Place processes at startup, don’t reassign
  - Requires a priori knowledge

- Dynamic Balancing
  - Process-Migration
  - Adapts dynamically to changing loads

- Problems
  - Determination of current load
  - Distribution algorithm
  - Oscillation possible

- Successful in SMPs and clusters, not (yet ?) used in MPPs
- Most advanced dynamic load balancing: MosiX (next week!!)
Messages

- Hardware – Implementation (IBM SP 2)
- Object – Systems (RPC)
- Libraries with special operations (e.g. MPI)
- Active Messages
Active Messages

- **Goal:**
  - Very fast process communication over the network (latency)

- **Idea:**
  - Message contains address of procedure to be invoked
  - Message reception leads to procedure invocation (in analogy to interrupt handler)

- **Discussion**
  - Very successful: speed (CM5: 12 microseconds Round Trip)
  - But: repair of OS limitations?
MPI, very brief overview

- Library for message-oriented parallel programming.
- Programming-model:
  - MPI program is started on all processors
  - Static allocation of processes to CPUs.
  - Processes have “Rank”: 0 ... N-1
  - Each process can obtain its Rank (MPI_Comm_rank).
- Typed messages
- Communicator: collection of processes that can communicate, e.g., MPI_COMM_WORLD
- MPI_Spawn (MPI – 2)
  - Dynamically create and spread processes
MPI - Operation

- Init / Finalize

- MPI-Comm-Rank delivers “rank” of calling process, for example
  
  ```c
  MPI_Comm_Rank(MPI_COMM_WORLD, &my-rank)
  
  if (my_rank != 0 )
  ...
  else ....
  ```

- MPI_barrier(comm) blocks until all processes called it
- MPI_Comm_Size how many processes in comm
MPI – Operations Send, RCV

- **MPI_Send** (void* message, int count, MPI-Datatype, int dest, /*rank of destination process, in */ int tag, MPI_Comm comm) /* communicator*/

- **MPI_RCV** (void* message, int count, MPI-Datatype, int src, /* rank of source process, in */ int tag, /* can be MPI_ANY-SRC */ MPI_Comm comm, /* communicator*/ MPI_Status* status); /* source, tag, error*/
MPI – Operations Broadcast

- MPI_BCAST(
  void * message,
  int count,
  MPI-Datatype,
  int root,
  MPI_Comm comm)

- process with rank == root sends,
  all others receive message

- implementation optimized for particular interconnect
MPI – Operations

- Aggregation:
  - MPI_Reduce
    - Each process holds partial value,
    - All processes reduce partial values to final result
    - Store result in RcvAddress field of Root process
  - MPI_Scan
    - Combine partial results into n final results and store them in RcvAddress of all n processes
MPI - Operations

Compute: $a[0] \ a[1] \ a[2] \ a[3]$  

MPI reduce

Compute: $a[0] \ a[1] \ a[2] \ a[3]$  

MPI scan
MPI – Operations

- MPI_Reduce(
  void* operand, /* in*/
  void * result, /* out*/
  int count, /* in */
  MP_Datatype datatype,
  MPI_Op operator,
  int root,
  MPI_Comm comm)

predefined MPI_OPs:
sum, product, minimum, maximum,
logical ops, ...
Case Study for an SMP OS: K42

- Overview:
  - Supports “pile-of-work” style,
  - common/shared object (file) system,
  - processes have many threads
  - Threads of a process run on different CPUs and share address space
  - provides Linux syscall interface
- For more see http://domino.research.ibm.com/comm/research_projects.nsf/pages/k42.index.html and paper in Eurosys 2006
Case Study: K42

- Overview:
  - Invented and/or aggressively explored many new interesting techniques (not in this lecture):
    - To support user-level thread scheduling within processes (address spaces)
    - Hot Swapping of components/implementations
    - Based on their own microkernel
  - Objective:
    - migrate invented techniques into main stream Linux
  - This lecture concentrates on K42's “clustered objects”
Clustered objects (CO) in K42

Key ideas
- Minimize sharing, maximize locality
- Avoid global data structures and locks
- Hide internal distribution/implementation structure
- Per-processor “representatives” and one “root” as central entity
CO example: counter

Operations: “inc” and “getval”
alternatives: global variable ./ local reps & root counter cases

- Inc frequent, getval infrequent: local rep better
- Getval frequent, inc infrequent: shared global variable better
  (in K42: swapping even at run time)

Next few slides taken from Jonathan Appavoo with permission ...
Common MPP Operating-System-Model

- PE: compute intensive part of application
  - Micro-Kernel
  - Start + Synchronization of Application
  - elementary Memory Management (no demand paging)

- all other OS functionality on separate Servers or dedicated nodes
- strict space sharing:
  only one application active per partition at a time
Space Sharing

- Assign partition from field of PEs
  - Applications are pair wise isolated
  - Applications self responsible for PEs
  - shared segments for processes within partition (Cray)
- Problems:
  - debugging (relatively long stop-times)
  - Long-running jobs block shorter jobs
- Isolation of application with respect to:
  - Security
  - Efficiency
- Buzzword: “eliminate the OS from the critical path"
Space Sharing

- Hardware-Supported assignment of nodes to applications
- Partitions
  - static at configuration
    Installed by operator for longer period of time
  - Variable(Blue Gene/L):
    Selections and setup on start of Job established by “scheduler”
  - Very flexible (not in any MPP I know):
    - increase and shrink during operation
    - Applications need to deal with varying CPU numbers
Case Study: IBM – Blue Gene/L

- Applications:
  - Storm prediction
  - Protein folding
  - ...
- Requirements:
  - Fold large protein:
    - 1 Year computation on Petaflop computer
  - Ranking 1 in Top 500
    using 16384 compute nodes (possible 65536)
IBM – Blue Gene Hardware

© IBM Research
Functional HW organisation

- compute nodes: applications only, up to 65 536
- i/o nodes: IO, file system interaction, ... up to 1024
- separate file servers
- service nodes

compute and IO-Nodes: newly developed, identical
Compute and IO Nodes

- 2 32 bit PowerPC 700 MH with two 64 bit FPU each
- L1 cache (not coherent),
  2KB L2 and 4MB L3 cache (coherent)
  L2 hit 10 cycles, L3 hit 25 cycles, L3 miss 75 cycles
- some SRAM buffers for intranode communication
- GB ethernet,
- torus, tree, interrupt/barrier control
  (additional ASIC for cross plane communication)
- very low power
IBM – Blue Gene Interconnect

- 5 Networks:
  - 3D Torus 64 (cabinet) x 32 (midplane) x 32 (cpu card) only used by compute nodes memory mapped: “local injection FIFO” and “local reception FIFO” can be partitioned 175 MB/sec per link
IBM – Blue Gene Interconnect

- 5 Networks:
  - Tree Network
    used Combine and Broadcast (network contains ALUs)
    interaction with IO/Nodes
  - Barrier + Interrupt Network
  - GBIT Ethernet to JTAG (machine control)
  - GBIT Ethernet to outside
    (communication with other systems, file servers)
Broadcast and Combine Tree Network

MPI: root

network nodes contain ALU

ca 2 microseconds latency for complete tree
IBM – Blue Gene Interconnect

- “psets”:
  - can be assembled as needed (with restrictions)
  - e.g.: 1:8, 1:128, 1:1024 (compute:io)
System Software Overview

Figure 1

High-level architectural view of a complete Blue Gene/L system.
System Software: Overview

Figure 2
High-level view of the Blue Gene/L system software architecture.
Principle Decisions

- strict space sharing
  one application per partition at a time
  enables user-mode communication without protection problems
- one application thread per compute node
- no demand paging:
  to avoid page faults and TLB misses
System Software

- compute node kernel: newly developed (CNK)
- io-nodes: Linux
- front end: compilers etc
- overall: MPI exploiting the communication hardware of compute nodes
- service node: control of the whole machine, DB2
Node and Roles

User App
File API
messages

NFS Client Service Processes
ION-Kernel
messages

NFS GPFS ...

CNK
ION
FS

Distributed Operating systems, K42 and Blue Gene
Hermann Härtig
IO-Nodes

- all outside communication of compute nodes goes thru IO-Nodes
- Job launch and control for their “psets”
- provide system services needed by application
- no user application code on IO-Nodes
- diskless; boot image via RAM-Disk
- only one of the two CPUs are used by Linux (L1 cache not coherent)
- CIOD (control and io daemon): program launch, signaling, termination, IO
  point to point messaging with compute nodes
Compute Node Kernel

- single-user dual threaded minimal kernel
- flat fixed-size 512MB address space
- kernel protected using MMU
- physical resources are partitioned between user and kernel
- access to torus from user mode
- glibc runtime; no fork exec etc
  IO shipped to IO-Nodes
Compute Node Kernel

Messages, three layers

- HAL packet: delivery of packets
- Messages: arbitrary size, reordering
- MPI
  - Specific support for reduction, broadcast, processes
  - Packets and messages: “active messages”
Compute Node Kernel, 2 modes of op.

- coprocessor mode,
  dual threaded process sharing complete address space
  - One CPU:
    main application thread, non preemtable
  - Other CPU as "coprocessor":
    e.g., message passing services,
    but also computation in coroutine model
    co_start starts a computation
    co_join waits for completion
    all coherence must be handled by user program
- virtual node mode (next slide)
Compute Node Kernel, 2 modes of op.

- coprocessor mode
- virtual node mode
  2 single threaded processes, bound to CPUs
  - each process has access to half of the memory
  - share access to communication
  - can communicate only via message passing
Service Node

Core Management and Control System (CMCS), acts as global OS:

- makes all long-term policy decisions
- in coop with IO-Nodes performs/controls system management:
  monitoring, booting, temperature control, configuration registers
Job Execution: Init and Boot

- partition allocation: identify set of unused nodes
- compute “personality” for each node
  - view of torus etc
- boot image for
  - CNK ca 128KB
  - IO-Node ca 0.5 MB + RAM Disk
- configuration info loaded into “Personality Area” of each node
- IO-Node mounts file systems etc as needed
Summary

- Workstations, Server and Supercomputers with low number of CPUs: State of the Art
- Symmetric Parallel Computers with X-CPUs: Successful application at OLTP
- Massively Parallel Computers X-hundred to x-thousand CPUs: promising (since a long time ... )
References

- Dave Culler, Jaswinder Sing, Parallel Computer Architecture – A HW/SW-Approach 99, Morgan Kaufmann

- Kai Hwang, Zhiwei Xu, Scalable Parallel Computing 97, McGraw-Hill

- Curt Schimmel, Unix Systems for Modern Architectures 94, Addison Wesley
References

- Blue Gene/L programming and operating environment

  IBM Journal of Research and Development
  Volume 49, Number 2/3, Page 367 (2005)

- MPI Tutorial