Scalable Parallel Systems

Contributions 1990-2000

Marc Snir
Topics

- SP – from research to product
- Programming models & software for scalable parallel systems
- Evolution toward shared memory
- The future of High Performance Computing platforms (Blue Gene, and beyond)
Research not Covered by Talk

- K42 – a scalable, customizable, 64 bit optimized Linux kernel
- Parallel datamining
- Parallel job scheduling
- Java for high performance numerical computing (compiler optimizations, libraries, Java standard modifications)
  - [www.javagrande.org](http://www.javagrande.org)
- UTE – scalable trace visualization tool
  - Supercomputing 00
Prehistory: Vulcan (90-92)

- Hardware architecture for 1 TF/s MPP
  - Up to 32K I860 nodes
  - Multistage packet switching network (50 MB/s link)
  - Specialized compute nodes, I/O nodes, general purpose nodes
  - Option for mirroring
  - Simple communication interface (in/out buffers; interrupts at low/high watermark)
    - blocking sends & polling receives
- Hardware architect: Monty Denneau
- Software architect: Marc Snir
Theory of INs – was it of any use?

- Used multi-stage network (theoretically optimal)
- But used bidirectional topology (log₄N stages, rather than log₈N – 50% more)
  - Done to simplify packaging and initialization
- Used theoretical framework for deadlock analysis
- Did not use theory about randomized routing
  - But developed, after the case, a new theory to explain why this was the right choice.
Randomized routing

- Some permutations can take up to $\sqrt{N}$ steps.
- All permutations can be done in $O(\log N)$ steps if one uses randomized routing in a network with $2\log N$ stages and (Valiant, Upfal,...).
- Less than $2\log N$ stages will not do (Valiant).
- Practice (simulation) “bad permutations” do not seem to happen.
  - Hence we built a network with very limited redundancy.
- Theory (Upfal, Pelegrin, Snir): an extra $\log \log N$ stages are sufficient if, first, nodes numbers are scrambled randomly.
  - After the case justification!
Vulcan Operating Environment

- Special purpose – focus on large parallel computations
  - General services provided, via proxies, at general purpose nodes
- No multitasking – use space partitioning
- Node services:
  - Nanokernel for interrupt handling, thread scheduling, packet handling
  - Thread, message-passing and I/O libraries
  - User threads
- Global services:
  - Parallel file system (Vesta)
Vesta (initial) design

- High I/O bandwidth: files striped across all I/O nodes
  - User control of striping (similar to partitioning of 2D array in HPF)
  - Support for partitioning and linear addressing of partitions
- Efficient fine grain file access at comp nodes, assuming coarse grain at I/O node
- Fast switch: no need for caching at compute nodes
- Protected comm: kernel code at I/O node, user library at compute node
- Accesses are atomic (serializable)
- Support for concurrent (lazy) checkpoint
From Vulcan research to SP product

- IBM decided to make “Vulcan-like” product; we dropped Vulcan work (even though hardware was coming online!) and worked feverishly on product
  - ✗ No publications and no glory for Vulcan work
  - ✓ No need to complete “obsolete” project
- Needed more general purpose system
  - Can be used as modest cluster/farm, not only large supercomputer
  - Does not need front-end system
- Needed to reuse IBM technology
  - Reduce unique development and sell more IBM iron
SP

Work started Feb 92
First product Sept 93
> 1B$/year sale in last few years

ASCI White
- 512 16w SMP nodes
- 12.28 Teraflop/s peak performance
- 160 Terabyte disk storage
Main changes from Vulcan

- **I860 small card node**
  - RS/6000 W/S drawer node
    - ✓ Blue, more memory, more I/O (more general purpose)
    - ✗ Bigger node, smaller max size
    - ✗ No mirroring (smaller max size)
    - ✗ Switch attached via I/O bus – higher latency
      - Need nonblocking comm and separate comm controller

- **Special nanokernel**

- **RS/6000 W/S drawer node**
  - ✓ Blue, more memory, more I/O (more general purpose)
  - ✗ Bigger node, smaller max size
  - ✗ No mirroring (smaller max size)
  - ✗ Switch attached via I/O bus – higher latency

- **General Unix (AIX) O/S at each node**
  - ✓ Blue, less unique development, more general purpose, better fit for farm/cluster apps
  - ✗ Less application control of scheduling & resource management
    - Need coarser grain apps & nonblocking comm
Vesta modifications

comp node
- App proc
- Vesta lib

Switch

comp node
- App proc
- Vesta lib

I/O node
- kernel
  - Buffer Cache
  - Virtual Disk

I/O node
- kernel
  - Buffer Cache
  - Virtual Disk
Vesta modifications

- Kernel code at compute node (because of multitasking, sharing, and lack of user-kernel communication support)
- Caching at compute nodes (because of slower communication)
  - Requires coherence protocol
  - Forces large blocks, to amortize overheads

Status:
- Vesta became product (94?)
- replaced by more conventional file system (GPFS) in 97 (?)
MPI-IO, to get back performance

- Parallel user library
  - Uses fast user space messaging
  - Manages partitioning and data distributions

- Status:
  - MPI-IO part of SP parallel sw & is optimized for GPFS
Parallel programming models

- **Driving principles:**
  - Program at as high level as possible; but
  - Parallelism (control distribution, load balancing) and locality (data distribution, communication) must be handled (sometimes/often/always) algorithmically

- **High road:**
  - Global view of computation: one program mapped (by compiler) over entire system
    - Programmer guides mapping via directives/annotations

- **Low road (machine view):**
  - Fixed number of processes (= nodes), each possibly multithreaded, that communicate via messages
High Road: High Performance Fortran

High Performance Fortran
- Standard Fortran + directives (static) for data distribution
- Compiler splits computation (parallel loops) according to data distribution

Limited success:
- Limitations of model (static, regular data decompositions; data parallel model)
- Limitations of implementations (mapped atop relatively inefficient communication layer, for portability; limited investment in optimizing compilers)
Low Road: MPI

Goals:

✓ Standard
✓ Precise semantics: ordering, progress

Efficiency: many opportunities for early binding

- Predefined groups, predefined datatypes, predefined communication channels
- Implementations do not take advantage of those!

✓ Good support for modular design
  - communicators

MPI is a very successful standard; like any successful standard, it has slowed down innovation
Sources of inefficiency in MPI

- High software overhead, due to layered implementation and to generality
  - Could be avoided via early binding (offline or online compilation)
  - Requires global analysis (future work?)
- Overheads inherent in MPI communication semantics
- Lack of adequate hardware support
Multiple copying

- No direct processor access to adapter space in user mode (protection)
- No DMA from pageable memory
- Limited DMA intelligence (no scatter/gather, no send-recv matching)
Complex protocol (long message)

- Need more powerful adapter
- Need simpler communication programming interface
Simpler interface: RMC

- Single sided operations: put, get, update
  - Separate communication from synchronization
  - Avoid need for complex matching engine
- Implemented efficiently atop SP hw (LAPI)
- Added (later & imperfectly) to MPI

- Also need remote enqueue
  - Multiple (remote) writers, single (local) reader queue
  - Needed for efficient implementation of messaging
FAST: hardware engine for RMC

- Command FIFOs
- Command Buffer
- Address translation & routing tables
- Virtual cache coherent DMA engine
- Output Engine
- Input Engine
- FIFOs
- memory bus
Virtual DMA – address translation

- Locally enforced protection
- Virtualization of nodes
  - Enables migration and multiplexing
- Efficient support for many-to-one (client-server) and asymmetric communication patterns
- No need to pin memory
  - Pager has to call communication agent at pageout!

local addr
space id

remote mem
window id
logical
addr

local node
lookup

route
remote
table #
logical
addr

remote node
address
translation
physical
addr

Put (laddr, rwinid, raddr, len)
FAST status (1996)

- Simulated (in software)
  - AIX modified to work with FAST
  - RMC user interface developed
    - software overhead < 2 µsec
  - MPI implemented atop FAST
    - x4 reduction in software overhead for short messages!
- FAST project canned in favor of FASTER project
  - development unwilling to invest for latency reduction
  - strong interest in support for coherent shared memory, in addition to messaging
- Sorry, yet again no publications (but a few patents)
Why shared memory?

- Message passing:
  - Software scheduled computation, software scheduled communication & synchronization
- Coherent shared memory:
  - Software scheduled computation, software scheduled synchronization, on demand communication

😀 Shared memory: simpler programming model, lower software overhead

😢 Shared memory: more complex hardware, no latency hiding

- Idea: support coherent shared memory; enable software controlled data movement, for latency hiding

- Message passing used for performance optimization, but not needed for correctness
FASTER = FAST + snoop

- Critical timing
- Large directories

Communication operations implicitly started upon snoop match
PRISM (aka FASTER) Goals

- No changes in SMP node architecture
- Few changes in (AIX) Operating System
- Scales to hundreds of SMP nodes (thousands of processors)
- Does not (necessarily) introduce single points of global failures
- Supports selective memory sharing across nodes and across OS images
Conventional (CC-NUMA) design: single OS controls multiple nodes; conventional address translation logic; all addresses managed by adapter are real & global.
General address translation mechanism across OS images

client node

private memory  NUMA memory  COMA memory

address translation

local real memory  phantom memory  local real memory

redirection

home node

local real memory  local real memory

coherence
Memory management

- Added prefetch and prestore commands, for COMA segments

- Unix V like global shared memory services
  - allocate & attach of global shared segment effected by explicit commands
    - at load time (for global addressing) or dynamically
  - home pages allocated upon page faults
  - coherence effected by hardware
PRISM adapter block design

- Dataflow architecture style
- Used high-level protocol verification tools
Status

- Software completed
  - global job management
  - global memory management
  - run-time library
  - compiler prototype
- Paper published with initial results (HPCA 98)
  - high level hardware architecture done
- Project stopped/converted
  - funding problem
  - schedule problem: to be relevant for product, needed to target next product generation; this required significant architecture changes (no exposed memory bus)
- Derived design will appear soon in IBM products
Industrial research (IBM)

- Can undertake large projects, with diversified team
  - Typically, 4-6 people; rarely, >15
- Hard to bring large, good projects to their (academic) conclusion
  - Need to adapt to quick changes in product map
  - If technology is promising then team is preempted by development
- **The IBM Research dilemma:**
  - Research one step ahead of development steamroller is often steamrolled
  - Research far ahead of development steamroller is perceived as irrelevant and is hard to sell to development, even when successful
Future of High-Performance Computing Architecture

- High-performance systems have moved backward, in terms of good communication & synchronization support and ease of programming
  - lure of commodity based supercomputing seems irresistible
- But “commodity” hardware in the future includes many new types of devices, such as embedded memory, embedded microprocessor cores, and network processors
  - support better communication (to memory, to other nodes)
  - support message-driven computation
- > 95% of silicon area and power consumption of modern systems goes to storage and communication
  - need to focus programming model, compiler optimizations, architecture design, etc. on communication, not computation
- ?? message-driven microarchitecture, software controlled communication & synchronization
Future of High-Performance System Software

- Need to develop parallel system services
  - Global OS services and parallel application system interfaces
- Need to develop a model and framework for parallel interfaces between parallel application components
  - collective parallel method invocation
Main contributors

- Peter Corbett
- Eknath
- Hubertus Franke
- Mark Giampapa
- Joefon Jann
- Beng Hong Lim
- Pratap Pattnaik
The End

http://www.research.ibm.com/people/s/snir