Tempest: A Substrate for Portable Parallel Programs*

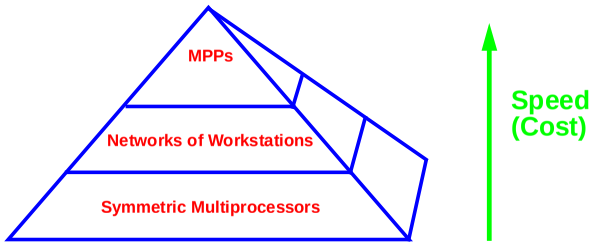
Mark Hill, James Larus, David Wood
Wisconsin Wind Tunnel Project
University of Wisconsin
http://www.cs.wisc.edu/~wwt

Satish Chandra, Trishul Chilimbi, Babak Falsafi, Rahmat Hyder, Alvin R. Lebeck, Mike Litzkow, Shubu Mukherjee, Rob Pfile, Steven Reinhardt, Brad Richards, Anne Rogers, Yannis Schoinas, Eric Schnarr, Steve Swartz, and Guhan Viswanathan

*. Sponsors: ARPA (B550, Wright Labs #F33615-94-1-1525), NSF (MIP-9225097 & CCR-9101035), NSF PYI/NYI (MIP-8957278, CCR-9157366, & CCR-9357779), DOE (DE-FG02-93ER25176), UW Grad School, WARF, Bell Labs, DEC, Sun, TMC & Xerox

Can Parallel Computing Become Ubiquitous?

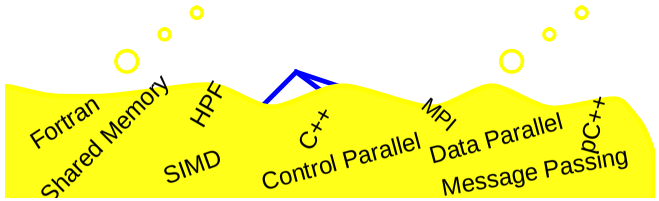
- Parallel hardware pyramid:



- Wide range of cost/performance trade-offs

Not Unless Parallel Software Improves

- Parallel software morass:

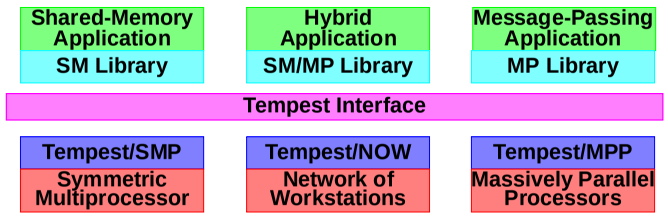


- Disparate programming models & languages
- Abysmal portability

Parallel Software

- Compare with uniprocessor software
  - Diverse programming models & languages
  - Most computers support most models & languages (portability)
- How should parallel computers portably support?
  - Coherent shared memory
  - Distributed computing
  - Fine-grain message passing
  - Coarse-grain message passing
  - Data parallelism
  - Hybrid models

Tempest: A Substrate for Portable Parallel Programs



- Tempest provides mechanisms
  - Communication, naming, & caching
- User-level code (libraries, compilers, programmers) implements policies
  - Transparent shared memory, custom protocols, message passing, ...
- Diverse implementations
  - SMPs, NOWs, and MPPs

Using Tempest

- Start with transparent shared memory
  - Library linked with application
  - Develop algorithms and application
- Find performance bottlenecks
  - Typically, communication of a few key data structures
- Exploit customized shared memory
  - Use custom protocols or message passing
  - Libraries of protocols
- Tempest = parallel communication “assembly language”

Outline

- Motivation
- Tempest Interface
  - Active messages
  - Bulk Data Transfer
  - Virtual memory control
  - Fine-grain access control
- Using Tempest
- Implementing Tempest

Active Messages & Bulk Data Transfer

- Active messages
  - Small messages that invoke user-level handlers (like Berkeley)
  - Exchange control & small data transfers
- Bulk data transfer
  - Large messages (e.g., CM-5 channels)
  - Higher bandwidth, but higher latency
- Virtual Memory Mechanisms
  - User-level control of segment’s address space
  - Page-based distributed shared memory (Ivy & Munin)

Fine-Grain Access Control

- Tag for fine-grained (32–128 bytes) memory blocks
- Program registers action handlers
- Check tag before LOADs and STOREs

	Invalid	Read-Only	Read-Write
LOAD	Action		
STORE	Action	Action	

```
if not access_valid (address) then
  invoke_action (pc, address);
LOAD_or_STORE
```

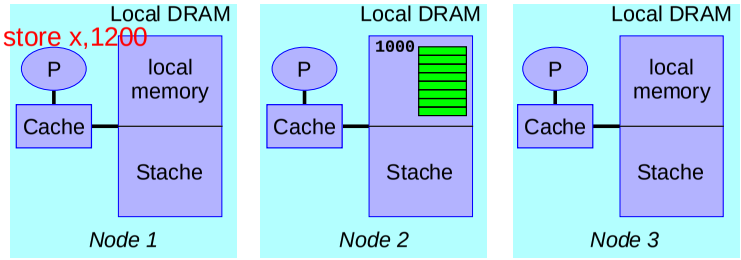

Fine-Grain Access Control, cont'd

- Key innovation in Tempest
  - Shared-memory hardware already has it
- Allows efficient, sequentially-consistent shared memory
  - Page-based DSM suffer from false sharing
  - Stache: fine-grain distributed shared memory
- Custom shared memory
  - Protocol software runs at user-level
  - Application-specific coherence protocols
    - Update (not invalidations) for select data structures
    - New memory semantics
    - Fine-grain copy-on-write for data parallelism

Outline

- Motivation
- Tempest Interface
- Tempest Applications
  - Stache Transparent Shared Memory
  - Message-Passing Protocols
  - Custom Protocols
- Tempest Implementations

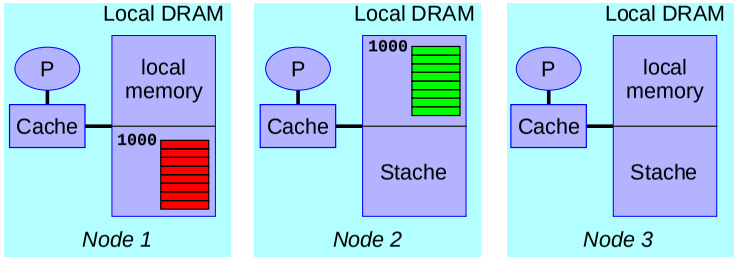
Stache: Transparent Shared Memory*



- Each node has local memory and Stache
- Node 2 is home for page 1000
- Node 1 writes block in page 1000

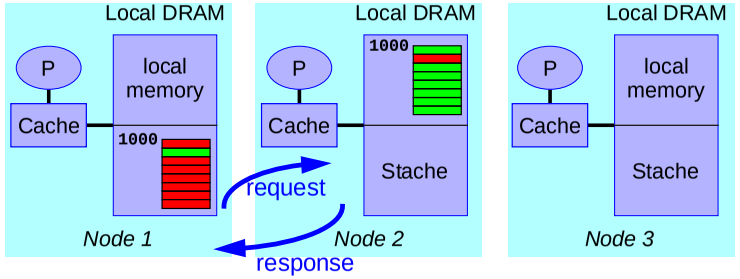
*. Reinhardt et al, ISCA94

Stache, cont.



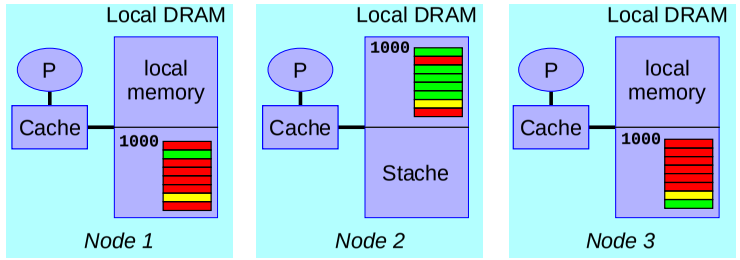
- Access causes page fault (VM mechanism)
- Page fault vectored to user handler (VM)
- Handler maps page and retries access (VM)

Stache, cont.



- Access fails and invokes action handler (fine-grain access control)
- User-level protocol runs (active messages)
  - Handler sends request message to home node
  - Home sends data block
- Install block and retry, successfully

Stache Summary



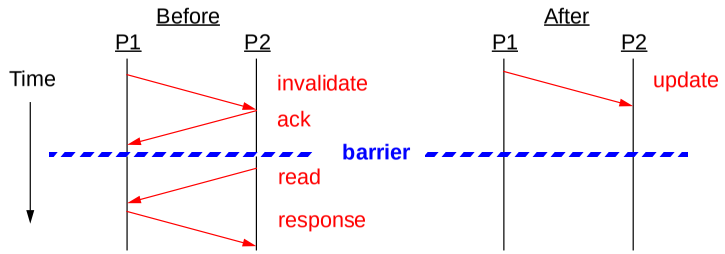
- Allocation on pages (VM mechanisms)
- Coherence on blocks (fine-grain access control)

Message-Passing Protocols

- Program can communicate with messages as well
  - Active messages and bulk transfer
- More efficient transfer when communication pattern understood
  - Use in place of shared memory for key data structures
- Does not exploit
  - Virtual memory
  - Fine-grain access control

Application-Specific Protocol (EM3D)*

- EM3D has producer-consumer communication

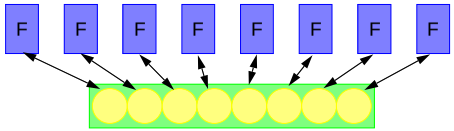


- Inefficient with invalidation protocol (4 messages)
- Add delayed updates for select data structures to Stache
- Minimal messages for known communication pattern

*. Schoinas et al, SC94

Language-Specific Protocol (LCM)*

- Support C**—large-grain data-parallel language
  - Apply parallel function to aggregate
  - Function invocations execute simultaneously & instantaneously



- Fine-grain, copy-on-write memory system
  - Loosely-Coherent Memory (LCM)
- Very different semantics from transparent shared memory
  - Language-specific

*. Larus et al, ASPLOS6

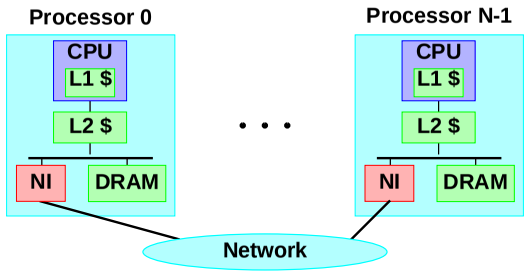
Tempest Summary

- Tempest provides mechanisms
  - Communication, naming, & caching
- Policies in user-level software (libraries, compilers, programmers)
  - Don't pay for policies you don't use
  - No expectation of bug-free code
- Possible policies (not part of Tempest!)
  - Stache transparent shared memory
  - Message-passing protocols
  - Custom coherence protocols

Outline

- Motivation
- Tempest Interface
- Using Tempest
- Implementing Tempest
  - Fine-Grain Access Control
  - Artifacts and Performance

Future Parallel Computers



- Nodes built from commodity components
  - One or more processors, caches, memory
- Custom components possible
  - Network and network interface (NI)
  - Fine-grain access control hardware

System Support for Tempest

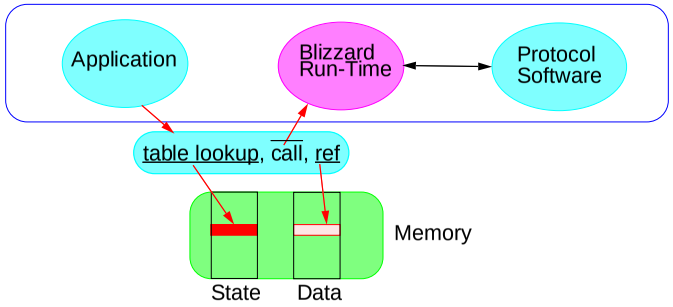
- Messaging
  - send message
  - invoke handler at receiver (PC in message)
  - send/receive memory blocks
- Virtual address translation
  - standard TLB sufficient
- Fine-grain access control (next slide)

⇒ Handlers run user-level software

Implementing Fine-Grain Access Control

- Must
  - manipulate tags
  - detect access faults
  - invoke access fault handler
- Want
  - Low-end: no special hardware
  - High-end: high-performance
- Implementation alternatives
  - Executable editing
  - Error correcting codes (ECC)
  - Bus snooping hardware

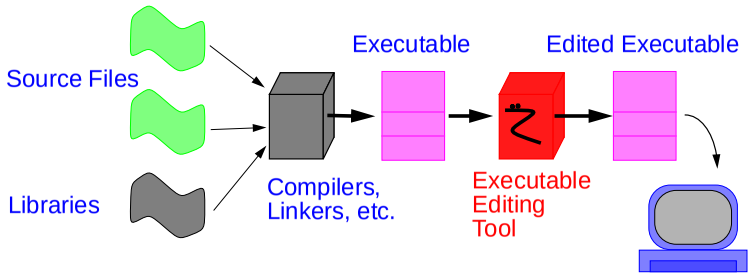
Executable Editing



- Rewrite executable to insert memory-state test
  - Portable
  - Only necessary before possibly-shared LOADs & STOREs
  - 8-instruction overhead (SPARC)

EEL: Executable Editing*

- Add foreign code and modify existing code in executable program



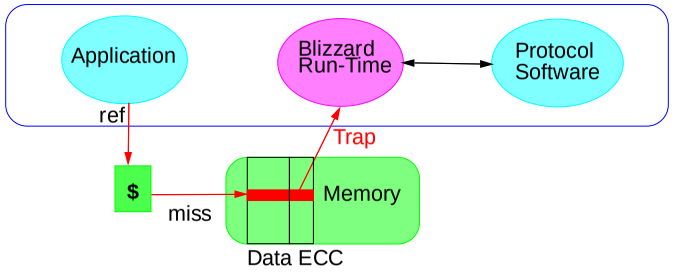
+System tools unchanged

+Expose complete process (including libraries) without source code

+Convenient to use

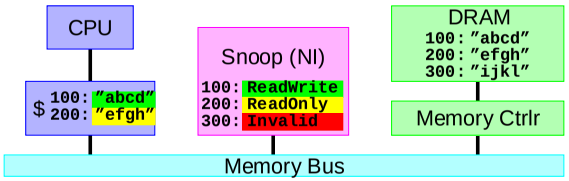
*. Larus & Schnarr, PLDI '95

ECC Bits



- Mark invalid blocks by setting double-bit ECC error
  - No overhead on no-action case
  - Requires restartable ECC exceptions & kernel modifications
  - Action slower due to trap overhead

Bus Snooping Hardware

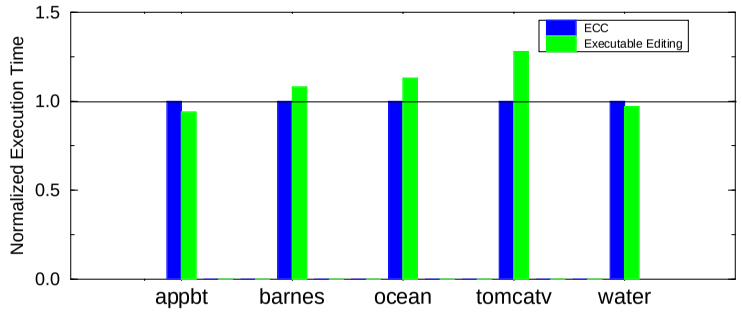


- Snoop hardware validates access
  - Logically checks tag at each LOAD & STORE
  - Actual checks only on bus transactions
    - Uses bus coherence operations to control data in CPU's cache
  - No processor or cache modifications
    - Snoop must be able to lock out CPU and invoke actions

Tempest Implementations

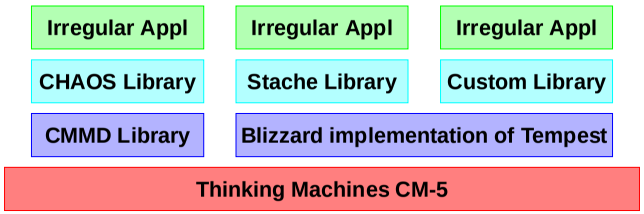
- CM-5 (Blizzard)
  - 64-node Thinking Machines CM-5
  - Uses
    - Executable editing
    - ECC bits
- Typhoon MPP
  - Proposed & simulatied
  - Bus snooping in custom NI chip
- COW (Cluster Of Workstations)
  - 40 dual-processor Sun SparcStations w/ MyriNet (soon)
  - Uses
    - Executable editing
    - ECC bits
    - Bus snooping hardware (called Typhoon-0 or T0)

Executable Editing vs. ECC on CM-5



- Stache w/ 128 byte blocks on 32-node CM-5
- Executable editing works well (assuring portability)

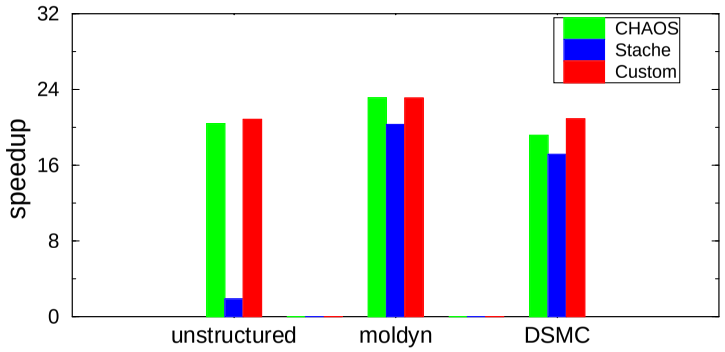
Custom Protocols on 32-Node CM-5*



- Compare shared memory, custom protocols, and domain-specific library
- CHAOS is library for irregular programs on message-passing machines
- Blizzard/CM-5 using ECC bits for fine-grain access control

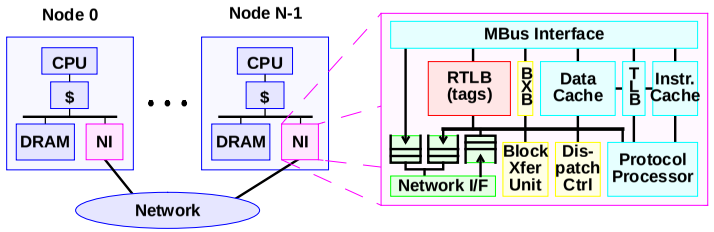
*. Mukherjee et al, PPOPP95

Custom Protocols on CM-5



- Custom protocols perform similar to CHAOS on 32-node CM-5
- Tempest supports a wider class of applications

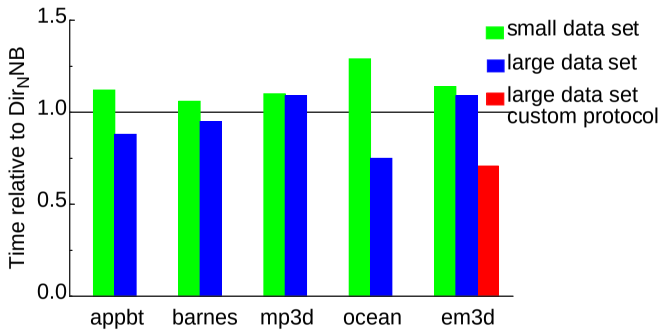
Typhoon



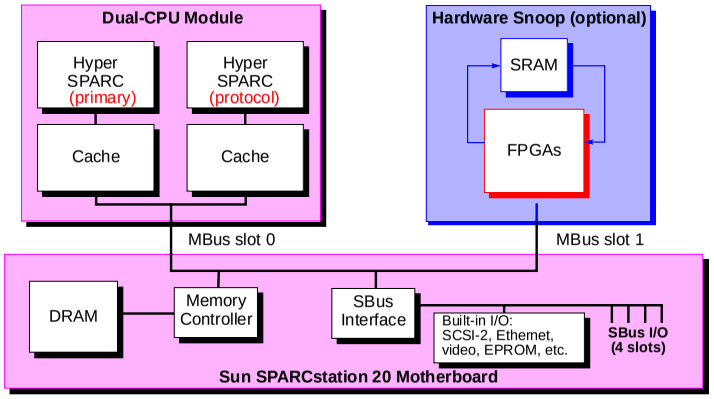
- RTLB snoops to provide fine-grain access control
  - per-page entries: tag vector, virtual address, more
- support for block send/receive, tag downgrade
- hardware handler dispatch
- integrated processor has single-cycle access to NI, RTLB

Typhoon vs. CC-NUMA [Reinhardt 94]

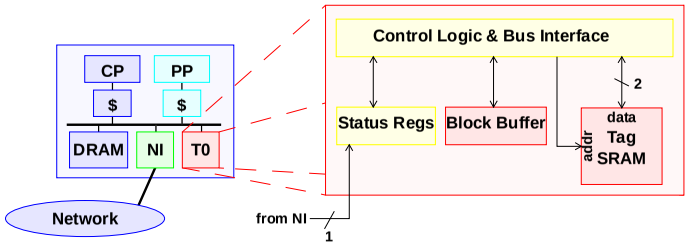
- Execution time relative to all-hardware Dir_NNB system
- 32 nodes, 256K HW caches, DASH network parameters



Wisconsin COW Node



Typhoon-0



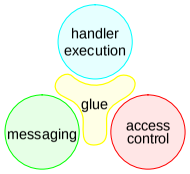
- off-the-shelf protocol CPU, network interface
- custom fine-grain access control support
- implemented for COW (MBus SS-20s)

Typhoon-0 Features

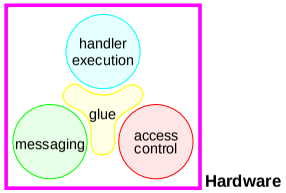
- Fine-grain access control
  - two-bit-wide SRAM array
  - shadow space for user access
  - reverse translation etc. in software
- Handler dispatch
  - cacheable dispatch vector register
  - burst transfer of access fault information
  - single bit from NI indicates message arrival
- Performance numbers coming soon

Implementing Fine-Grain Access Control (Summary)

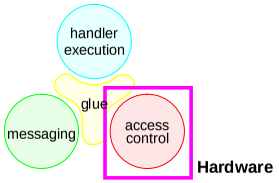
CM-5: No HW



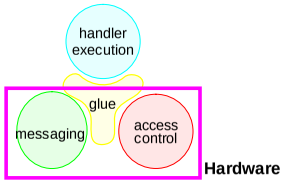
Typhoon: All HW



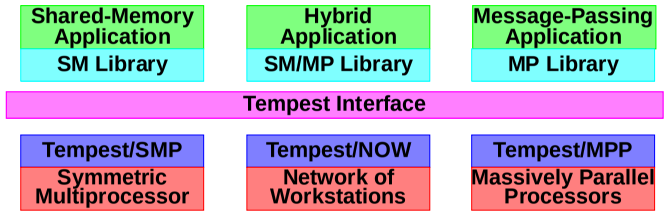
Typhoon-0: Simple HW



What Next?



Tempest Summary



- Tempest provides mechanisms
  - Communication, naming, & caching
- User-level code (libraries, compilers, programmers) implements policies
  - Transparent shared memory, custom protocols, message passing, ...
- Diverse implementations
  - SMPs, NOWs, and MPPs

For More Information

WWW: `http://www.cs.wisc.edu/~wwt`
FTP: `ftp ftp.cs.wisc.edu; cd wwt`
EMAIL: `wwt@cs.wisc.edu`

- Everything: Compcon95 (Hill et al.) & Annotated Bibliography
- Tempest Interface: Spec (Reinhardt) & ISCA94 (Reinhardt et al)
- Tempest Protocols
  - Stache: ISCA94 (Reinhardt et al)
  - LCM: ASPLOS6 (Larus et al)
  - Custom: SC94 (Falsafi et al) & PPOPP95 (Mukherjee et al)
- Tempest Implementations
  - Typhoon: ISCA94 (Reinhardt et al)
  - Blizzard/CM-5: ASPLOS6 (Schoinas et al)
  - Blizzard/COW: none yet

Some Related Work

	software runs on...	SW on cache side handles...	SW at directory handles...
Typhoon	embedded user-level CPU	remote misses in DRAM cache	everything
Typhoon-0	general-purpose user-level CPU	remote misses in DRAM cache	everything
Alewife (MIT)	primary CPU	nothing	hardware overflow cases
FLASH (Stanford)	embedded system controller	every hardware cache miss	everything
Start-NG (MIT)	dedicated CPU	every hardware cache miss to shareable data	everything