A Simulation Environment for Microprocessor Development

Chien Chen, Leslie A. Barnes and
John R. Slice

HAL Computer Systems, Inc.

Microprocessor Division

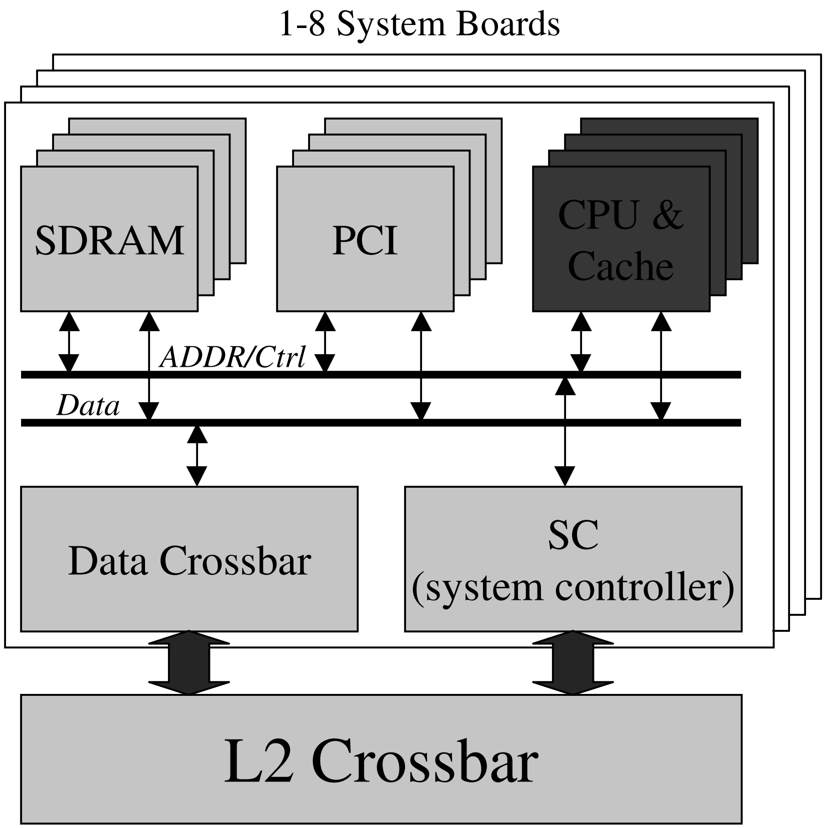
HAL MPD

- Principal design center for SPARC64[R] microprocessors
- Fujitsu high-end technical and commercial Unix server products
- SPARC64 V Design objectives include:
  - 1 GHz target frequency of operation
  - High performance (70+/130+ Spec95)
  - High reliability (RAS)

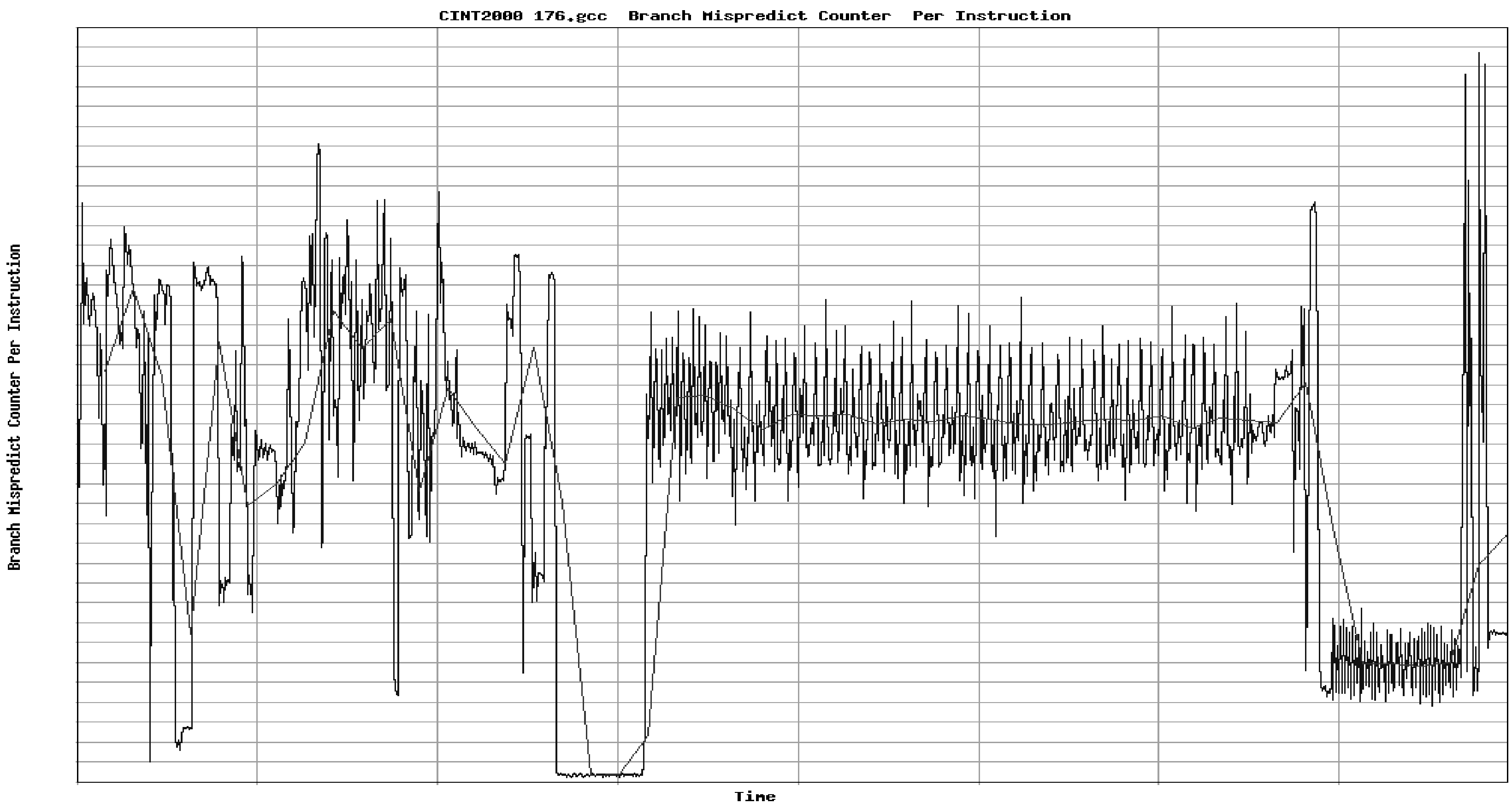
Performance Group Overview

- Simulation tools for microprocessor performance evaluation, for current and future processors
- Platform development done by FJ in Japan
  - we concentrate on cpu design
- Performance measurement for currently shipping products
  - our measurement work is aimed at gaining insight into real workloads and microprocessor issues for current designs
- Workload generation and analysis for simulation
  - SPEC CPU95 and CPU2000
  - TPC-C/TPC-D/TPC-H/TPC-R
  - Web/Java applications
  - Other FJ internal benchmarks

Example System Architecture



Measurement example: gcc branch mispredicts



SPARC64 V Microarchitecture Thumbnail

- Trace cache architecture
  - Can issue through two taken branches per clock
- 8-issue superscalar design:
  - Issue up to 6 instructions in Integer code, 8 in Floating Point
- Dynamic scheduling, super-speculative data distribution
- Hierarchical, large on-chip caches, external L3 cache support
- Extensive RAS support
- Complexity? What Complexity?
  - "Daunting complexity" (Microprocessor Report)

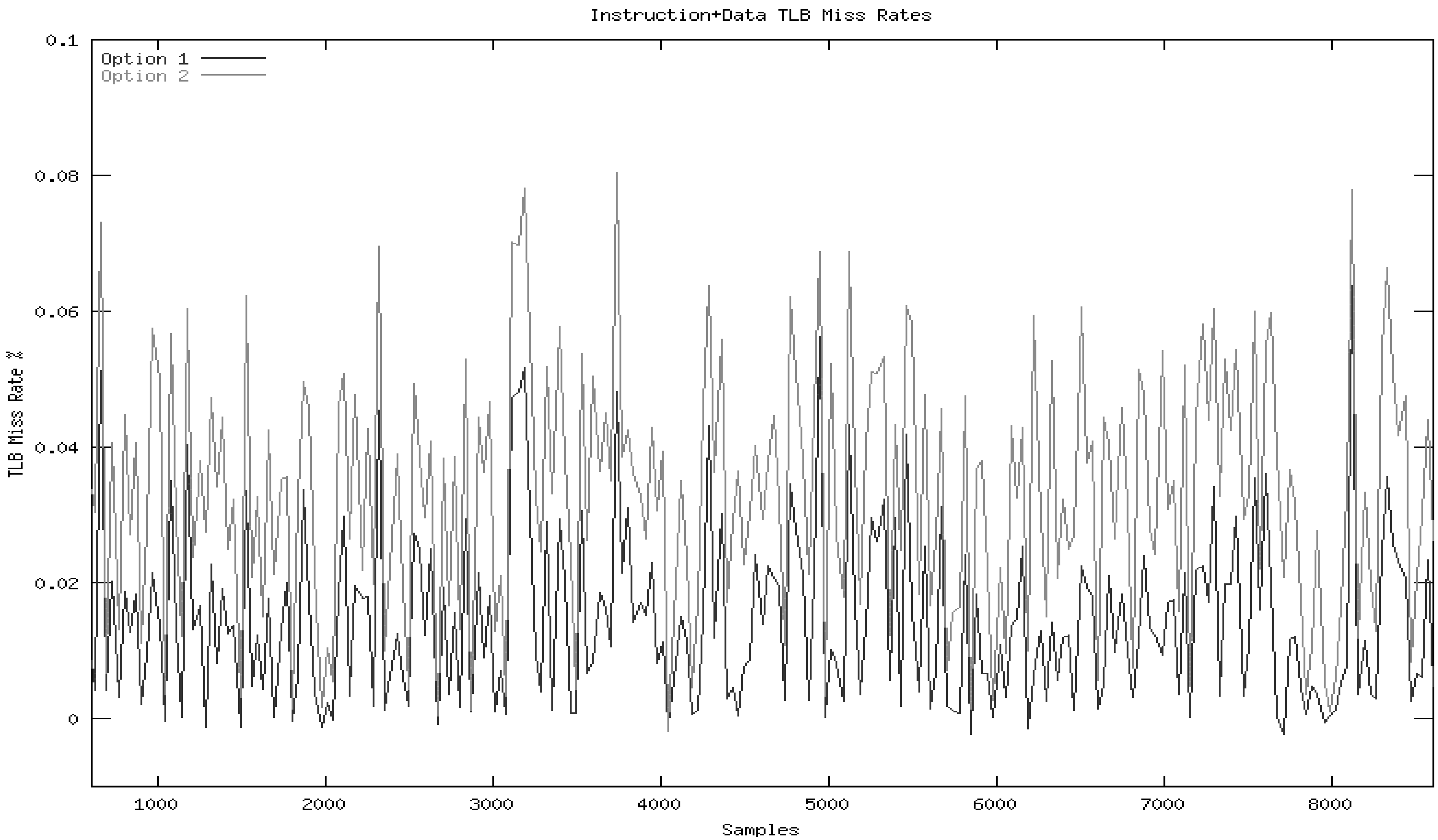
Halsim

- a fast SPARC V9 architecture and platform simulator
- emulates a variety of MMU’s from different V9 implementations (including USII and GP7000F)
- has a platform model, which emulates FJ GP7000F platforms, with SCSI, ethernet, serial chips, pseudo terminal, PCI and SBUS support, mouse/keyboard
- can boot unmodified Solaris 2.6/7, including firmware, in MP mode
  - does require working OBP and Solaris for a given platform
- can be configured on the network just like a real machine
- used for firmware/kernel development work and performance analysis
  - gdb and analyzer interface to cpu

Halsim (ctd)

- can be used to generate traces or analyze memory access behaviors
  - detailed information on I-Fetch, D-Fetch and D-Store activity
- captures all operating system and device driver activity
- runs large database and scientific applications unmodified
- supports checkpoint/restart (aka slamming)
  - register/memory/disk state saved to disk
  - can be reloaded (slammed) to restart as if nothing happened
  - very important for performance analysis
- does not provide timing information
- contains an interface which can be used by other simulators, such as those giving accurate timing information for complex microarchitectures

Halsim (ctd)



Instruction+Data TLB Miss Rates
Option 1
Option 2
TLB Miss Rate %
0.1
0.08
0.06
0.04
0.02
0
1000
2000
3000
4000
5000
6000
7000
8000
Samples

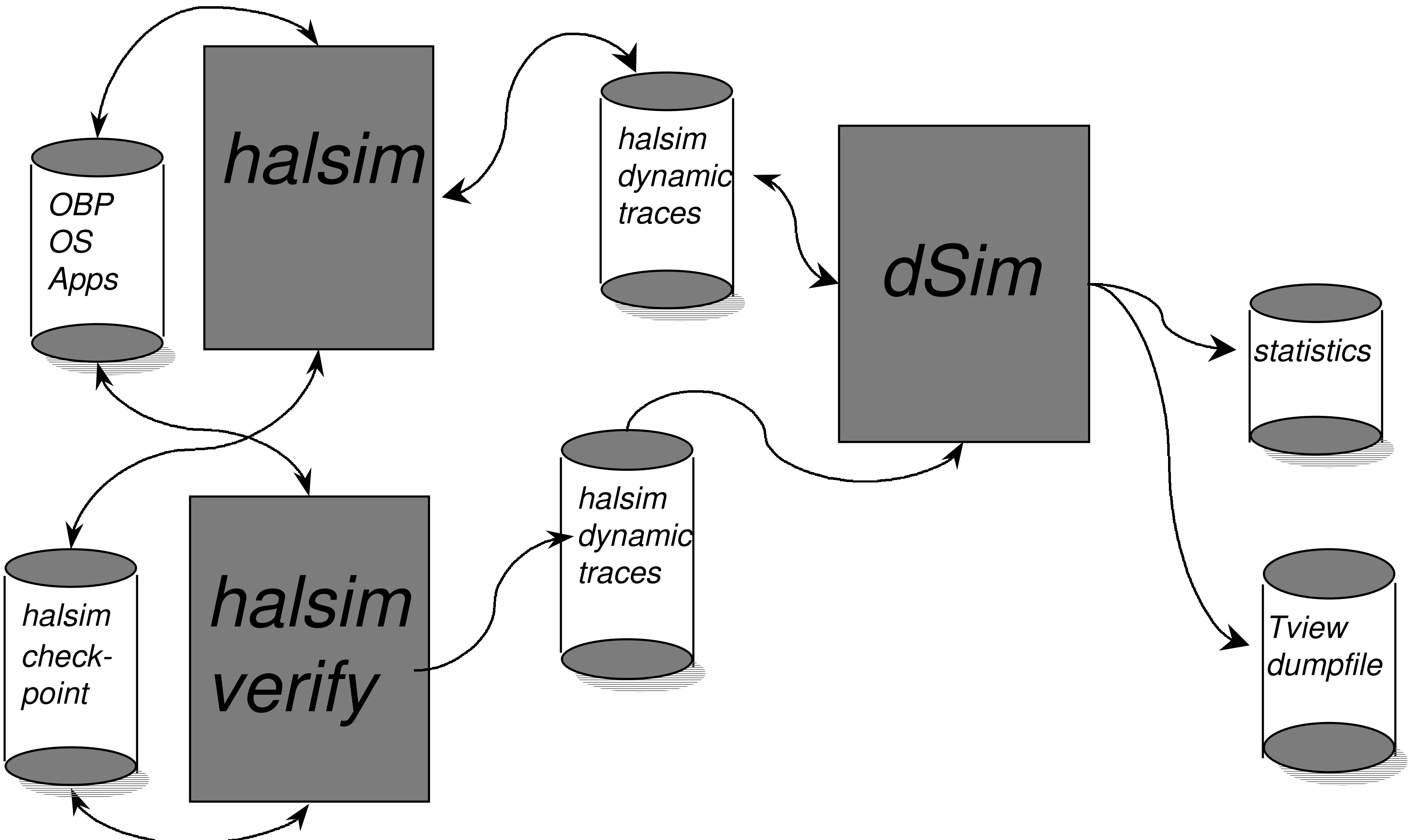
dSim: SPARC64 V microarchitecture simulator

- Trace-driven Simulator
  - Written in C++
  - SPARC64 V micro-architecture
  - Accurate
  - Easy to configure to answer what-if questions
- Models Speculative Effects
- Not an Execution Model
  - but has some execution model characteristics

dSim Features (ctd)

- How dSim models speculative effects
  - by default, halsim is always on “correct” execution path
  - halsim extensions
    - allow external agent (dSim) to modify its state (cc, pc, registers)
    - checkpoint its internal states such that re-execution with different branch directions is allowed
    - data (registers/memory contents) can be transferred to dSim for certain “execution model” like behavior
  - dSim actions
    - at each instruction boundary, tell halsim whether to continue, take a mispredicted branch, take a trap, or backup to a prior instruction.
  - halsim verify option ensures correct instruction commit stream and address/data stream

dSim Simulation Flow



Pros and cons of “dynamic trace-driven” approach

- Pros:
  - get platform and operating system support immediately, able to start running larger applications
    - with an execution driven model, it takes a lot more effort to get to this point
  - trace-driven is more tolerant to modeling error and more self-correcting (still need “verify” though for QA purposes)
  - easier to change the model and evaluate many different microarchitecture features
- Cons:
  - less overlap with rtl model, harder to compare in detail
  - trace-driven is more tolerant to modeling error and more self-correcting

What was not modeled properly?

- dSim is fundamentally trace driven. Data speculation was not completely modeled (control speculation was modeled properly)
  - we did model load value prediction, but not where data speculation affected control speculation. This was mainly a time limitation - eventually we determined that LVP was only useful if much more complexity was put into the processor. Since we had plenty of complexity already, we punted it.
  - Some aspects of super-speculative execution were only approximately treated. For example, bad speculative addresses.
  - Other data dependent operations were done properly (data dependent functional unit latencies etc).

What was hard about this?

- First step was to achieve firmware boot (OK prompt) with dSim in full speculative mode
  - felt fairly confident about simulator quality at this point
- Second step was OS boot with dSim (strips)
  - many problems encountered. Main problems were to do with instructions with side-effects (IO etc) on the speculative path, traps and interrupts
  - these need to be guarded on the speculative path
  - at very high trap levels, we limit speculation to avoid sending processor into “red mode”
  - kernel panicing was a sure sign we had problems
  - developing “verify” option was very valuable

What was hard about this? (ctd)

- Following this, we ran SPEC95 and other simple benchmarks, ironing out residual bugs
  - many billions of instructions run through halsim/dSim in full speculative mode
  - Found warmup was a very big problem with large memory latencies at 1GHz. Developed extensive warmup capability.
    - Halsim has a cache model, image is stored in checkpoint
    - This is slammed into dSim L3 cache
    - dSim has a "fast warmup" mode (exercise L3/D2/I2 caches only)
    - additional full warmup of simulator also done prior to statistics collection
  - At this point we were feeling pretty good
- Then we started our TPCC work

TPCC modelling

- Start with an experimental TPCC setup in our performance lab
  - 1-4 Tb of disk, SCSI raid/fiberchannel, 1-4 Gb memory
  - 1, 2, 4, 8 processor systems (GP7000F)
  - Oracle or some other database program (FJ internal)
- Tune this for reasonably optimal performance
  - Try to understand processor performance issues
  - we are not going to publish tpcc numbers with this setup
- Clone this onto halsim simulation environment
  - use standard system tools (sar/mpstat/vmstat etc) to ensure cpu utilization, interrupts, IO’s etc compare to our experimental setup
- Currently we are still uniprocessor with halsim/dSim
  - working on MP

What was hard about this? Epilogue

- TPCC stressed all aspects of the simulation environment severely
- Many bugs found
  - mostly in control interface between halsim and dSim, dealing with interrupts/IO's etc and in dSim itself.
  - several halsim bugs uncovered, some in interface code, others in mmu and various other corner cases
  - very difficult to debug due to large resource requirement needed to run the simulations
    - 1Gb or more of memory
    - large checkpoints (2Gb or more)
  - database programs soon tell you if database corrupted
- We never alter our original disks (read only). All changes are recorded in log files, so we cannot corrupt our original setup.

Results?

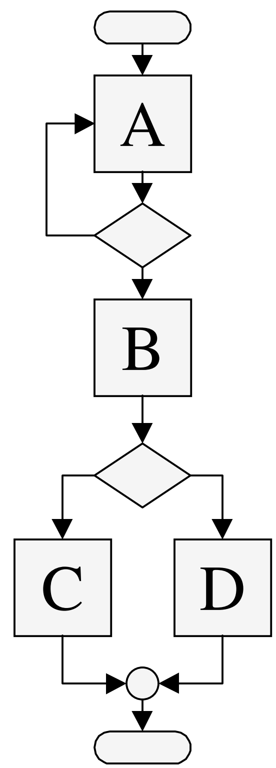
- Maybe next year …
- Thanks go to
  - Joe Fitzgerald/Mike Shebanow ($$$)
  - Steve McCann (TPCC/TPCD)
  - Sean Wang (halsim/ethersim/dSim)
  - David Billings (dSim)
  - Xiao-dan Yu (dSim)
  - Carol LePage (measurement/OS)

Backup slides after here

Trace Cache Architecture

- Decompose programs into *basic blocks* (BB) connected by control flow decisions
- SPARC64 V packs up to two BBs into a *packet*
  - Performed by Fill Unit
    - Pre-decode, intra-packet dependency resolution, and routing
  - Trace Cache (T$) caches output of Fill Unit

Program Flow Graph

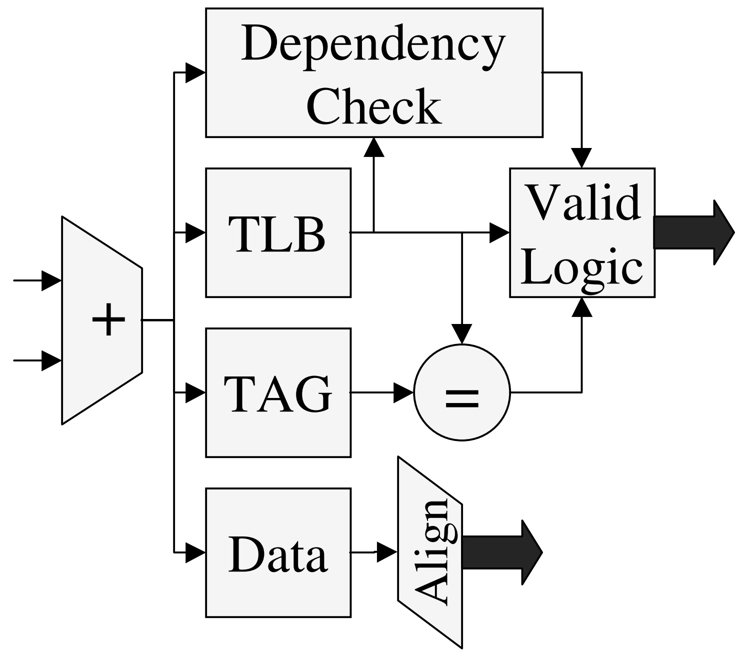


Dual-BB Trace

1	A	A
2	A	A
3	B	C

Super-Speculative Execution

- Example: L1D$
  - Time required to fetch data is less than time required to determine valid
- Super-speculative execution: distributing data to dependent function units prior to data valid
  - Assume valid and replay if wrong



Ethernet emulation: briefly

- Halsim provides full network capability, either between two halsim machines or halsim and a real machine
- Used “ethersim” which acts as an IP gateway connecting halsim subnet with Ethersim host subnet
- Ethersim snoops the host ethernet raw interface and sends packets directly via the same interface
- Functionally ok, but problems arise for performance modelling
  - simulated time vs real time (host ethernet is too fast for halsim). Halsim is interacting with real hosts and perceived ethernet bandwidth is much too high
  - checkpoint/restart difficult (cannot checkpoint external network state)
  - not very reproducible (other network activity impacts simulation)

Ethernet emulation: new scheme

- Working on a new scheme to address these issues
  - Record network activity via normal halsim run, in a logfile
  - Halsim will support recovering from a checkpoint and receiving packets from the logfile (virtual ethernet)
  - Filter out unrelated network activity
- Since the checkpoint/logfile are fixed, we can run multiple experiments with dSim under controlled conditions
- Working in parameterizable delays to allow incoming ethernet traffic from real hosts to be throttled down
  - on the timescale of halsim/dsim simulation (only a few hundred million instructions at a time), this should be ok

Verification: dSim accuracy matters

- Halsim is used for functional verification of rtl model
- We also do performance verification of rtl model with dSim
  - directed tests used to verify basic performance metrics for rtl and dSim
  - various small benchmark programs which can be run directly without OS are also used for performance verification
    - checkpoint slam capability needed for rtl model to do this properly (rtl is very, very slow, so need to sample different pieces of the benchmark via different halsim checkpoints).
  - Ultimately OS and real applications (at least, small instruction strips), will be done the same way