Virtual Memory Simulation Theorems

Mark A. Hillebrand, Wolfgang J. Paul

{mah,wjp}@cs.uni-sb.de

Saarland University, Saarbruecken, Germany

This work was partially supported by IBM

Overview

Theorem: the parallel user programs of a multiprocessor see a sequentially consistent virtual shared memory
(Correctness of multiprocessor hardware & part of OS)

Context

A (practical) approach for the complete formal verification of real computer systems:

1. Specify (precisely)
2. Construct (completely)
3. Mathematical correctness proof
4. Check correctness proof by computer
5. Automate approach (partially)

Example: Processor design

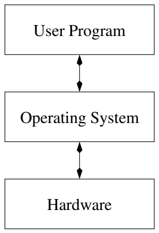
1.–3. [MP00]
Computer Architecture: Complexity and Correctness
Springer

4. [BJKLP03]
Functional verification of the VAMP processor
Charme ’03

5. PhD Thesis Project S. Tverdyshev
(Khabarovsk State Technical University)

Why Memory Management?

Layers of computer systems (all using local computation and communication):



! In memory management hardware and software are coupled extremely tightly.

DLX_V Configuration

A *virtual* processor configuration of DLX_V is a triple $c = (R, M, r)$:

- $R : \{\text{register names}\} \to \{0,1\}^{32}$
  where register names: $PC, GPR(r), status, \ldots$
- $M : \{\text{virtual memory addresses}\} \to \{0,1\}^8$
  where virtual memory addresses $\in \{0,1\}^{32}$
- $r : \{\text{virtual memory addresses}\} \to \mathcal{P}(\{\text{R}, \text{W}\})$
  where the rights R (read) and W (write) are identical for each page ($4K$).

! DLX_V is a basis for user programs.

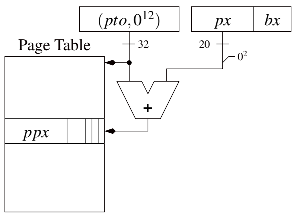
DLX_S Configuration

A real specification machine configuration of DLX_S is a triple $c_S = (R_S, PM, SM)$:

- $R_S \setminus R$:
  - $mode$ system mode (0) or user mode (1)
  - pto Page table origin
  - ptl Page table length (only for exceptions)
  - $emode$ exception mode
- PM physical memory
- SM swap memory

! DLX_S is hardware specification.

Page-Table Lookup



Let $c = (R_S, PM, SM)$.

- Virtual address $va = (px, bx)$

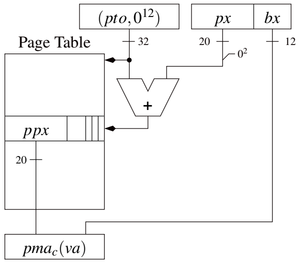
  px: page index
  bx: byte index

- $PT_c(px) = PM_4(\langle pto \rangle \cdot 4096 + 4 \cdot \langle px \rangle)$

31 – 12	11 – 3	2	1	0
$ppx[19:0]$	$\cdots$	r	w	v

$ppx_c(va)$: physical page index
$v_c(va)$: valid bit ($\leftrightarrow$ page in PM)

Address Translation



Let $c = (R_S, PM, SM)$.

- Virtual address $va = (px, bx)$

  px: page index
  bx: byte index

- $pma_c(va) = (ppx_c(va), bx)$
  pma_c: physical memory address
- To access swap memory, we also define:
  sma_c: swap memory address (e.g. $sbase_c + va$)

Instruction Execution

DLX_V uses virtual addresses:

- Fetch: $va = DPC$ (delayed PC)
- Effective address of load/store:
  $va = ea = GPR(RS1) + imm$ (register relative)

Hardware DLX_S for $mode = 1$ (user):

- If $v_c(va)$, use translated address $pma_c(va)$ instead of va.
- Otherwise, exception.
  (hardware supplies parameters for page fault handler)

Simulating DLX_V by DLX_S

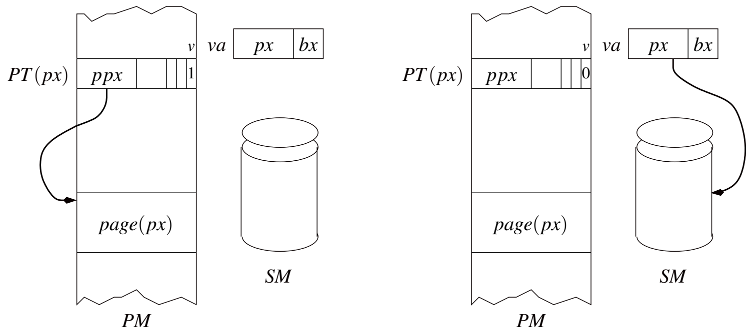
Let $c = (R_S, PM, SM)$ and $c_V = (R_V, VM, r)$.
Define a projection: $c_V = \Pi(c)$

- Identical register contents: $R_V(r) = R_S(r)$
- Rights in page table:

  $$\begin{aligned} \text{R} \in r(va) &\Leftrightarrow r_c(va) = 1 \\ \text{W} \in r(va) &\Leftrightarrow w_c(va) = 1 \end{aligned}$$

Simulating DLX_V by DLX_S

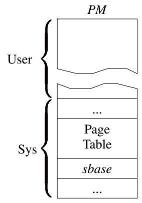
$$VM(va) = \begin{cases} PM(pma_c(va)) & \text{if } v_c(va) \\ SM(sma_c(va)) & \text{otherwise} \end{cases}$$



PM cache for virtual memory, PT is cache directory.
Handlers (almost!) work accordingly (select victim, write back to SM, swap in from SM)

Software Conditions

1. OS code and data structures (PT, $sbase$, free space) maintained in system area $Sys \subseteq PM$
2. OS does not destroy its code & data
3. User program (UP) cannot modify Sys (impossibility of hacking)
4. Writes to code section are separated from reads in code section by `sync` or (syncing) `rfe`



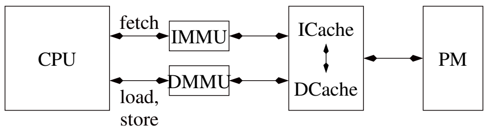
Standard `sync` empties pipelined or OoO (out of order) machine before next instruction is *issued*.

! Swap in code, then user mode fetch
= self-modification of code by OS & UP

Guaranteeing Software Conditions

1. Operating system construction
2. Operating system construction
3. No pages of Sys allocated via PT to UP
4. UP alone not self-modifying, handlers end with `rfe`

Hardware Implementation

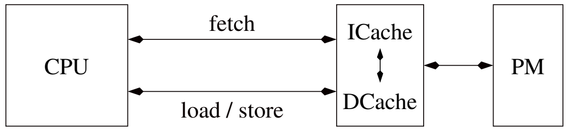


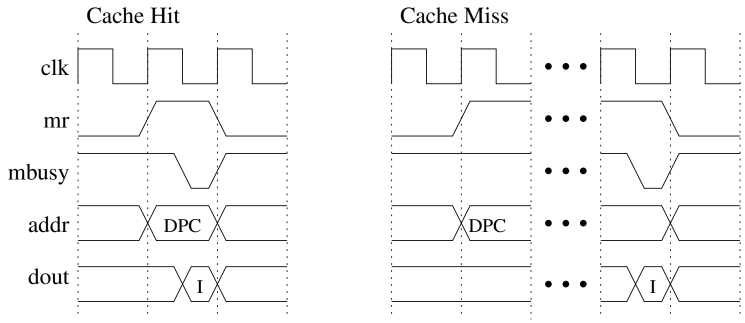
- Build hardware boxes MMU & a few gates
- Identify software conditions
- Show MMU translates if software conditions are met
- Show software meets conditions

! Then, almost done: establish simulation theorem.

Hardware I

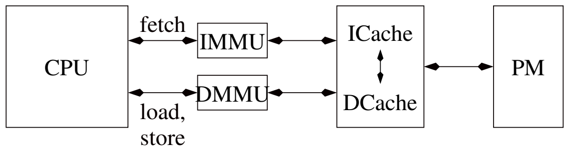
CPU – memory system – protocol





Hardware II

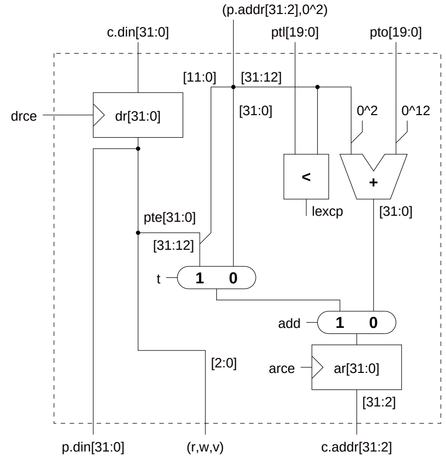
Inserting 2 MMUs:

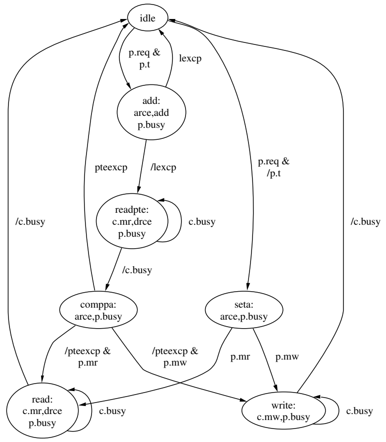


Must obey memory protocol at both sides!

Primitive MMU

Primitive MMU controlled by finite state diagram (FSD)





MMU Correctness

Local translation lemma:
Let T and V denote the start and end of a translated read request, no excp. Let $t \in \{T, \ldots, V\}$.

Hypothesis: the following 4 groups of inputs do not change in cycles t (i.e. $X^t = X^T$):

$G_1: \;\; va = p.addr^t, p.rd^t, p.wr^t, p.req^t$

$G_2: \;\; pto^t, ptl^t, mode^t$

$G_3: \;\; PT^t$

$G_4: \;\; PM^t(pma^t(va))$

Claim: $p.din^V = PM^T(pma^T(va))$
Proof: plain hardware correctness

Guaranteeing Hypotheses G_i

G_1 MMU keeps $p.busy$ active during translation

G_2 Extra gates: normal `sync` before issue not enough. If `rfe` or update to $\{mode, pto, ptl\}$ in issue stage, *stop* translation of fetch of next instruction.

G_3 User program cannot modify Sys. Preceding system code terminated (by `sync`)

G_4 Fetch: correct by `sync`
Load: assumes non-pipelined, in-order memory unit, extra arguments otherwise

Global Hardware Correctness (fetch)

Define scheduling functions $I(k,T) = i$: instruction I_i is in stage k during cycle T iff $(\ldots)$

- Similar to tag computation
- Key concept for hardware verification in SB

Hypothesis: $I(fetch, T) = i$, translation from T to V
Claim: $IR.din^V = PM^i_S(pma^i_S(DPC^i_S))$

Formal proof: part of PhD thesis project of I. Dalinger (Khabarovsk State Technical University)

Virtual Memory Simulation Theorem

Consider a computation of DLX_S:

Mode	$0 \ldots 0$	$1 \ldots 1$	$0 \ldots 0$	$1 \ldots 1$	$0 \ldots 0$
Conf.	$c_0 \ldots c_{\alpha-1}$	$c_\alpha \ldots c$	$c \ldots c$	$c \ldots c$	$c \ldots c$
Phase	Initialisation	User Program	Handler	User Program	Handler

Initialisation: $\Pi(c^\alpha_S) = c^0_V$

Simulation Step Theorem:

! 2 page faults per instruction possible (fetch & load/store)

Virtual Memory Simulation Theorem

Assume $\Pi(c^i_S) = c^j_V$.
Define:

- Same cycle or first cycle after handler:
$$s_1(i) = \begin{cases} i & \text{if } \neg pff^i \wedge \neg pfls^i \\ \min\{j > i, mode^j\} & \text{otherwise} \end{cases}$$
- Cycle after pagefault-free user mode step:
$$s_2(i) = \begin{cases} i+1 & \text{if } \neg pff^i \wedge \neg pfls^i \\ s_1(s_1(i))+1 & \text{otherwise} \end{cases}$$

Claim: $\Pi(c^{s_2(i)}_S) = c^{j+1}_V$

Liveness

We must maintain in Sys:
$MRSI$ (most recently swapped-in page)

Page fault handler must not evict page $MRSI$!

Formal proof: PhD thesis project of T. In der Rieden (Saarbrücken)

Translation Look-aside Buffers I

1-level lookup: formally caches for PT-region of PM

- Consistency of 4 caches:
  ICache, DCache, ITLB, DTLB
- Simply invalidate TLBs at mode switch to 1,
  sufficient by `sync` conditions

Translation Look-aside Buffers II

Multi-level lookup: TLB is *simplified* cache hardware:

- Normal cache entry:

c_ad	$v = 1$	tag	$PM(tag, c\_ad)$

- TLB entry:

c_ad	$v = 1$	tag	$pma^t(tag, c\_ad)$

  t : time of last `sync` / `rfe`
  Invalidate at mode switch to 1
- No writeback or load of lines
- Only ‘cache’ reads and writes of values $pma(va)$ by MMU

Formal verification trivial from verified cache

Multitasking with Sharing

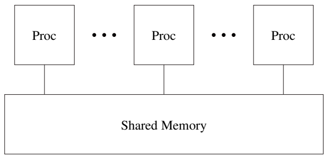
New virtual machine (user programming model!):

- Shared memory with logical addresses
- Tasks:
  - Processor configuration
  - Rights function
  - Abstract translation to express sharing, mapping virtual to logical addresses.
- Arbitrary (fair) scheduling

Simulation Theorem:

- Hardware part unchanged.
- Software part similar, more generalized (e.g. treatment of logical vs. physical sharing—COW)

Multiprocessor I



- PM: sequentially consistent shared memory (by cache coherence protocol)
- New software condition: before change of any page table entry *all* processors sync
- Sync hardware: very few circuits interfaced with processors
  Considered alone almost completely meaningless.

Multiprocessor II

Theorem: user programs see sequentially consistent virtual shared memory

Proof: Phases OS - UPs OS UPs

Global serial schedule:

- in each phase from sequential consistency of physical shared memory
- straightforward composition across phases
- remaining arguments not changed!

Summary

- Mathematical treatment of memory management
- Intricate combination of hardware and software considered
- Formalization under way