Présentation de sim-outorder

B. Goossens

Dali
Université de Perpignan Via Domitia

Plan de l'exposé

- A quoi sert sim-outorder?
- La structure de sim-outorder.
- Le pipeline de sim-outorder.
- Utiliser sim-outorder.
- Adapter sim-outorder.
- Conclusion.

Quand utiliser sim-outorder?

- **Micro-architecture.**
- **Architecture (nouvelle instruction).**
- **Profilage (chemin hors trace).**
- **Enseigner l'assembleur (avec Dlite!).**

Quand ne pas utiliser sim-outorder?

- Statistiques sur benchmarks (sim-profile).
- Performance de la hiérarchie mémoire
- (sim-cache, sim-cheetah).
- Performance de la prédiction de sauts
- (sim-bpred).

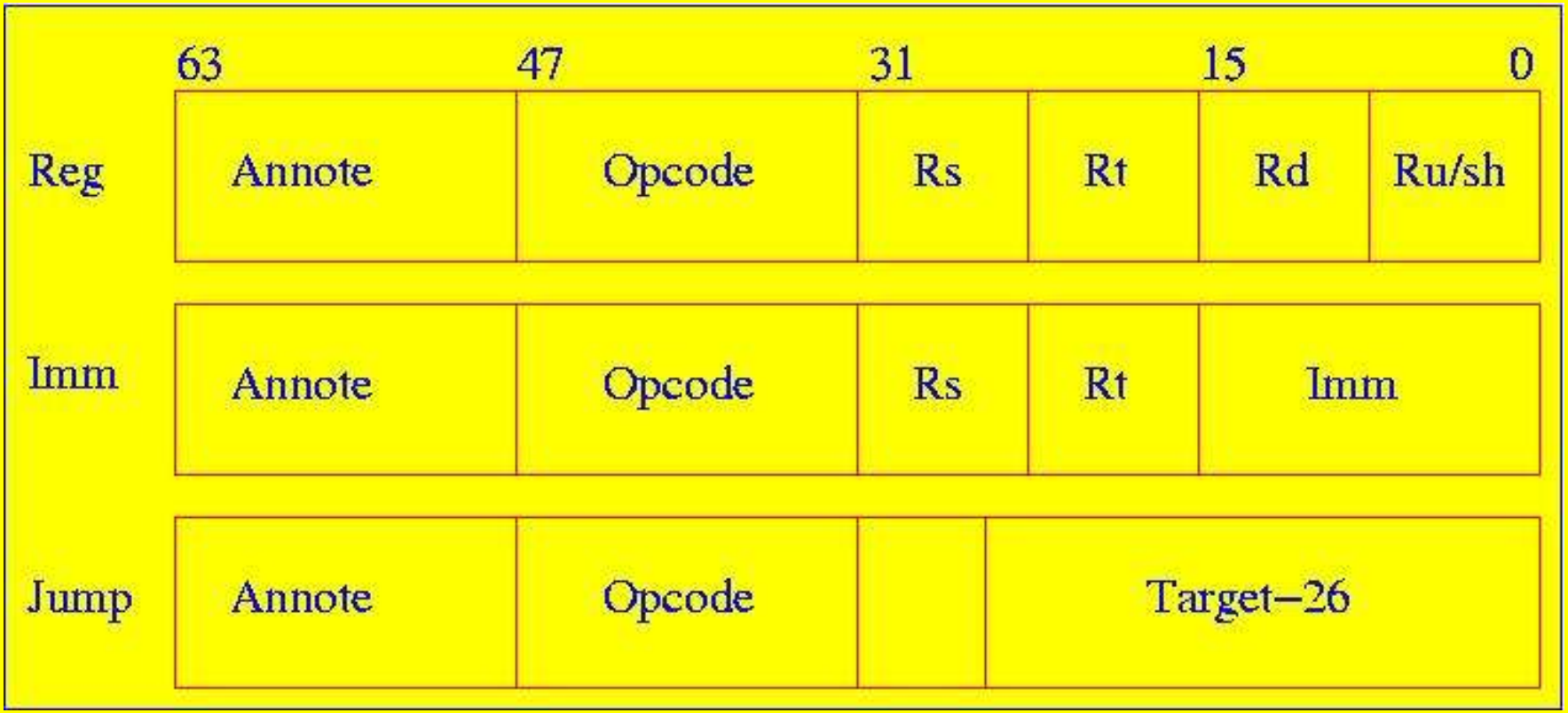
Composants du simulateur

- Un pipeline à 5 étages (f,d,i,(x),w,c)
- Une hiérarchie mémoire à 3 niveaux + tlb
- Un prédicteur de sauts paramétrable
- Des stations de réservation (# paramétrable)
- Une file des chargements/rangements

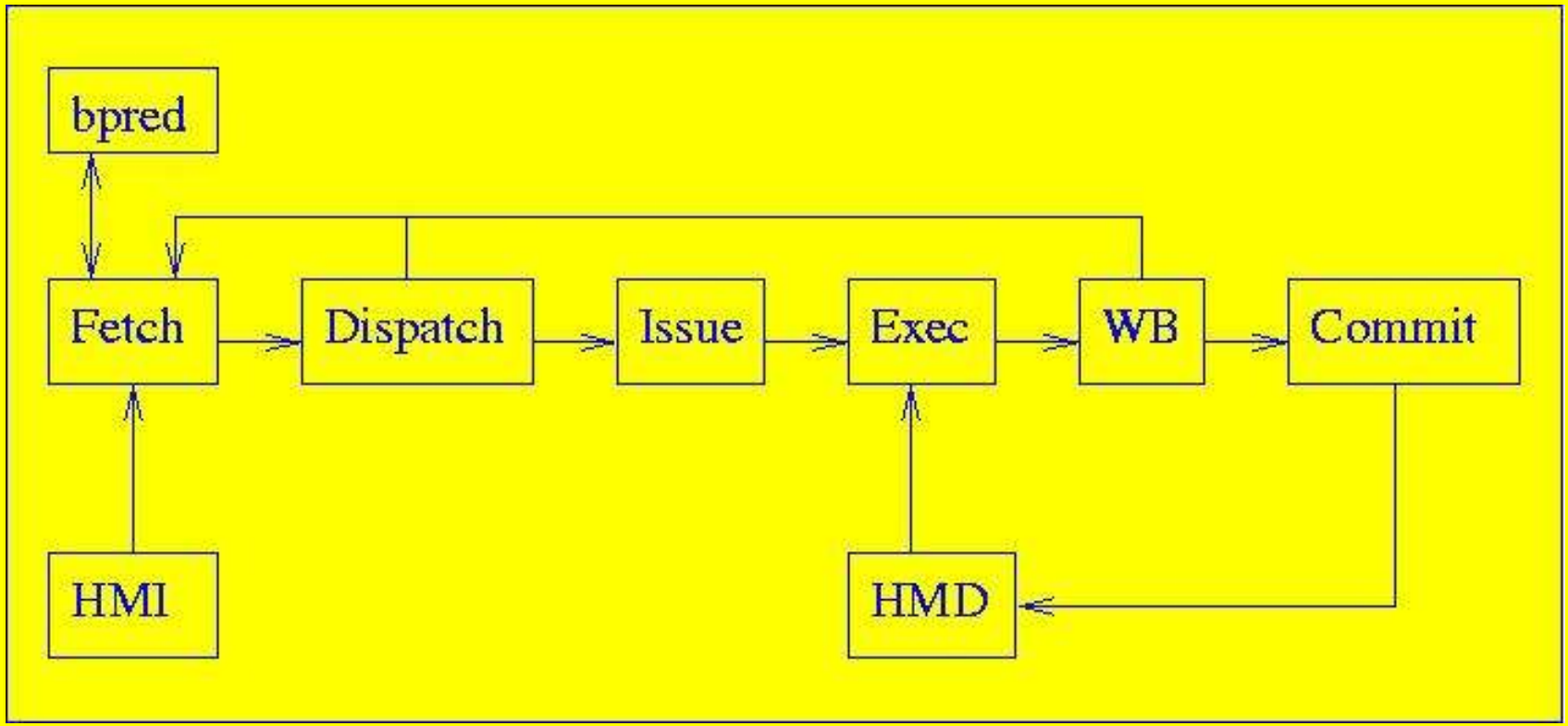
Architecture simulée

- Pisa (Portable ISA) et Alpha
- Pisa: instructions sur 64 bits
- MIPS IV-like
- 32 registres entiers, 32 registres flottants
- Appels systèmes par instruction SYSCALL

Architecture simulée



Micro-architecture simulée



La hiérarchie mémoire

- nom:#set:#asso:remplacement
- nom: il1, il2, dl1, dl2, itlb, dtlb
- Remplacement: lru, fifo, random
- Caches L2 séparés ou cache L2 unifié
- Latences des caches L1, L2 et TLB
- Latence mémoire premier mot et suivants
- Largeur du bus mémoire

Prédicteur de sauts

- Préditeur parfait
- Prédiction toujours pris
- Prédiction toujours non pris
- Prédiction bimodale compteurs 2 bits
- Prédiction à deux niveaux (histo + compteurs)
- GAg: 1 histo de w bits, 2^w compteurs 2 bits
- PAg: n histos de w bits, 2^w compteurs 2 bits
- Prédiction hybride (McFarling)

Prédicteur de sauts

- Le prédicteur est combiné avec le BTB et la RAS
- Il est consulté dans l'étage d'extraction
- Il peut être mis à jour en ordre ou spéculativement
- Mise à jour dans l'étage de validation
- Mise à jour dans l'étage d'écriture

Les stations de réservation

- La structure RUU est une file
- Les instructions entrent à l'extraction
- Les instructions sortent à la validation
- Sert de banc de registre de renommage
- Sert de point d'attente des sources
- Les registres architecturaux sont séparés

La file des chargements/rangements

- Les ld/st allouent une entrée en RUU (adresse)
- Les ld/st allouent une entrée en LSQ (donnée)
- Un chargement attend les rangements antérieurs
- Les rangements sont propagés à la LSQ
- La propagation se fait dans le cycle du rangement

Le pipeline

```
while (1) {
  commit();
  writeback();
  /* execute(); */
  issue();
  dispatch();
  if (!fetch_issue_delay) fetch();
}
```

L'étage d'extraction (fetch)

```
for (i=0; i<decode_width * fetch_speed
   && nb_fetched < ruu_size
   && !branch_fetched; i++) {
   fetch_inst(inst, mem, pc);
   if (fetch_issue_delay=imh_access(pc)) break;
   pred_npc=bpred_lookup(pc,inst);
   fetch_queue[tail]={inst,pc,pred_npc};
}
```

L'étage de distribution (dispatch)

```
while (i<decode_width * fetch_speed
   && nb_decoded < ruu_size && ls_dec < ls_size
   && nb_fetched!=0) {
   {inst,pc,pred_npc}=fetch_queue[head];
   MD_SET_OPCODE(op,inst);
   if (op&F_TRAP) break;
   set_rw_regs(op);/* DEFINST */
   run(op);/* spec run SYMCAT(OP,_IMPL)*/
   if (mis_predicted_target)
       fetch_issue_delay = branch_penalty;
```

L'étage de distribution (dispatch)

```
rs={inst,op,pc,npc,ppc};
if (MD_OP_FLAGS(op) & F_MEM) {
   lsq={inst,op,pc,npc,ppc,addr};
   set_dependency(o_rs,i_lsq);
}
set_dependency(o_head_create_vector,i_rs);
if (bpred_spec_update == dispatch)
   bpred_update(pc,npc,op);
```

L'étage de distribution (dispatch)

```
if (pred_dir != branch_dir) {
   spec_mode = TRUE;
   rs->recover = TRUE;
   recover_PC = npc;
  }
} /* dispatch next i */
```

L'étage de lancement (issue)

```
for (i=0; node && i<issue_width; next_node) {
   /* node: from ready queue */
   /* ready queue: no more dependent insts */
   if (get_fu(rs->op)) {
      rs->issued = TRUE;
      fix_result_latency(rs->op); /* op/mem */
      i++;
   }
}
```

L'étage d'écriture (writeback)

```
while (rs=completed_event) {
  /* list of completed ops */
  rs->completed = TRUE;
  if (rs->recover) {
    ruu_recover(); /* free rs */
    fetch_recover(); /* squash fetch */
    pc = recover_pc; spec_mode = FALSE;
    stack_recover();
    fetch_issue_delay = branch_penalty;
  }
```

L'étage d'écriture (writeback)

```
  if (bpred_spec_update == writeback)
    bpred_update(rs->pc,rs->npc,rs->op);
  if (spec_mode)
    broadcast_result(spec_create_vector);
  else
    broadcast_result(create_vector);
  update_ready_queue();
} /* next completed inst */
```

L'étage de validation (commit)

```
while (committed<commit_width) {
   if (!rs->completed) break;
   if (rs->op == ST) {
      if (!get_store_port()) break;
      dcache_write(); /* hidden mem lat */
   }
   if (bpred_spec_update == commit)
      bpred_update(rs->pc,rs->npc,rs->op);
   committed++;
   rs++;
}
```

Sim-outorder: la mise en oeuvre

Installation:

vi Makefile (choisir machine/système)

make config-pisa (ou config-alpha)

make

make sim-tests (teste tous les simulateurs)

Binaires exécutables:

sslittle-na-sstrix-gcc -g -O -o foo foo.c -lm

(ssbig ou sslittle) (little sur PC) (gcc ou f77)

Simulations:

sim-outorder -config f.cfg bench [args]

Sim-outorder: la mise en oeuvre

- Editer un fichier .cfg dans le répertoire config
- Exemple: default.cfg
- Taille des files
- Latence des sauts et des caches
- Type et taille du prédicteur
- Nombre d'unités fonctionnelles
- Nombre de ports dcache

Sim-outorder: la mise en oeuvre

./sim-outorder -config config/default.cfg
tests/bin.little/test-fmath
> sortie-test 2> mesure

Résultats sur la sortie erreur standard
récapitulatif de la configuration
sim_num_xxx (catégorie xxx validées)
sim_total_xxx (catégorie xxx exécutées)
IPC, caches, bpred

Sim-outorder: la mise en oeuvre

```
./sim-outorder -config config/default.cfg
      -ptrace trace.trc 100:500
      tests/bin.little/test-fmath
      > sortie-test 2> mesure
./pipeview.pl trace.trc | more
Trace des exécutions/validations
   @ 593
   xy = '0x409a20: lui      r8,0x7efe'
   [IF]  [DA]  [EX]  [WB]  [CT]
    xy
```

Adapter sim-outorder: ajouter des options

On peut ajouter des options de configuration

```
opt_reg_type(odb, "-mon:option", "emploi",
        &opt, valeur_init_type, TRUE, NULL);
```

A placer dans la fonction sim_reg_options

Déclarer static type opt dans simulator options

Apparait en tête de la sortie

Exemple:

```
opt_reg_int(odb, "-mon:option", "emploi",
        &opt, 0, TRUE, NULL);
```

```
-mon:option          0 # emploi
```

Adapter sim-outorder: ajouter des compteurs

On peut ajouter des compteurs

```
stat_reg_counter(sdb, "compteur", "emploi",
        &mon_cpt, valeur_init, NULL);
stat_reg_formula(sdb, "ma_formule", "emploi",
        "v1/v2*v3", NULL);
```

A placer dans la fonction sim_reg_stats

Déclarer static counter_t mon_cpt;

dans simulator stats

Apparait en sortie

```
  compteur          8631 # emploi
```

Adapter sim-outorder: ajouter des instructions

On peut ajouter des instructions annotées

```
asm("mul.s/a %0,%1,%2":"=f"(c):"f"(b),"f"(a));
```

On modifie sim-outorder pour traiter séparément les instructions annotées

```
if (!(inst.a & 0xffff0000)) /* traitement normal */
else /* new_id, new_od, new_sem */
```

Utiliser sim-outorder: exécution pas à pas

On peut exécuter en mode de déboguage

./sim-outorder -i bench

Exécution pas à pas

step, cont [a], break a, print [m] e, stats,
regs, fregs, dump a [n], dis a [n]

Conclusion

Sim-outorder est un outil fiable
Sim-outorder est très utilisé en recherche

Mais

Adapter sim-outorder est difficile
Les adaptations sont peu fiables
Adapter l'architecture est très difficile

Démonstration

Installation (simpleutils, simplesim, gcc)

Tests (simplesim, gcc)