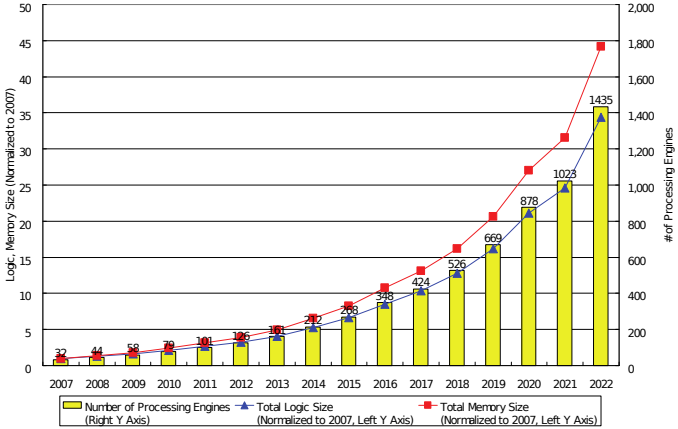
Simulation rapide des systèmes multiprocesseurs

Frédéric Pétrot, Laboratoire TIMA

tima.imag.fr/sls/people/petrot

frederic.petrot@univ-grenoble-alpes.fr

Tendances de l'intégration silicium



Feuille de route de l'ITRS [1] : nombre cœurs dans systèmes embarqués

1. International Roadmap on Semiconductors

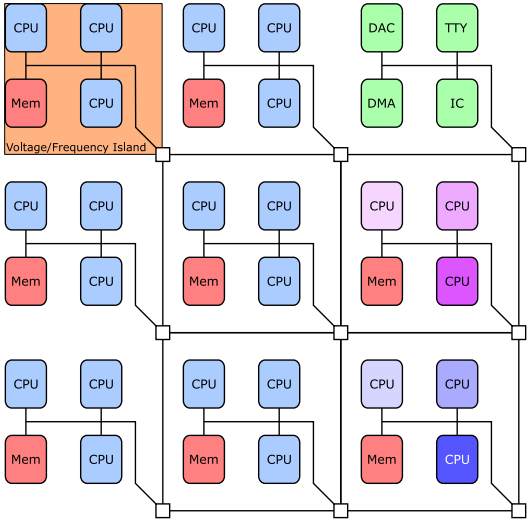
Architectures typiques de cette décennie

Caractéristiques

- massivement parallèle
- homogène ou hétérogène
- avec des hiérarchies mémoire souvent complexes

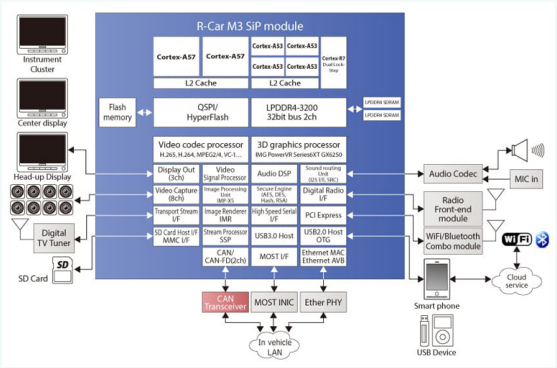
Propriétés

- flexible à très flexible
- fortes contraintes de coût
- fortes contraintes de temps de mise sur le marché

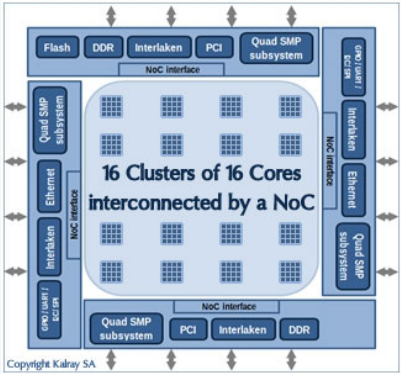


Architectures

Deux exemples industriels actuels



Renesas R-Car M3



Kalray

Plan

Simulation

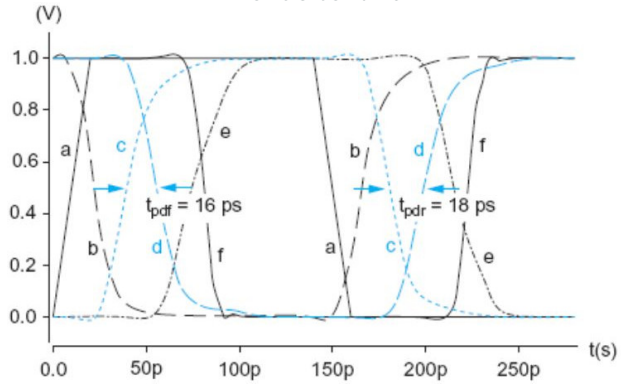
Une technologie qui couvre tous les aspects de la conception et de la validation des systèmes électroniques

Sujet de cette présentation

- simulation de systèmes électroniques numériques (et mixtes)
  - interconnectant de nombreux composants matériels
  - contenant un ou plusieurs processeurs
  - executant du logiciel
- plus abstrait que le niveau transfert de registres (RTL)
- focalisé sur exécution rapide du logiciel sur modèle de simulation du matériel

Simulation : niveaux d'abstraction

Monde continu



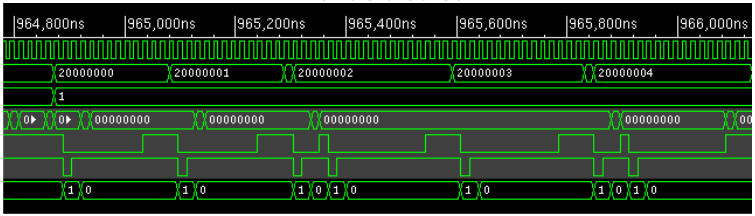
Résolution d'équations différentielles

(source *CMOS VLSI Design*, *N. Weste*)

Modélisation physique, pas (*step*) de simulation variables, extrêmement lent

Simulation : niveaux d'abstraction

Monde discret



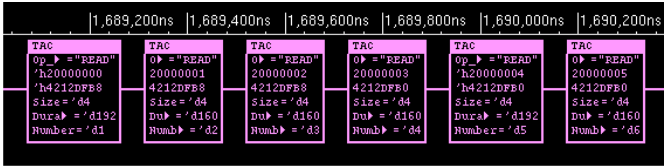
RTL (register transfer level), événements discrets, synthétisable

Communication par signal, pas fixe, δ-cycle après δ-cycle, très lent

CABA (cycle accurate bit accurate), à visée de simulation uniquement

Communication par signal, pas fixe, cycle après cycle, lent

Simulation : niveaux d'abstraction

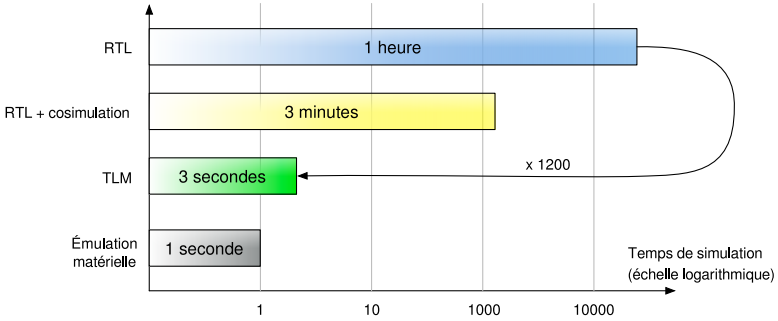


TLM (transaction level model), événements discrets, simulable

(source *STMicroelectronics*)

Communication par transactions gros grain, événements rares, rapide

Simulation : niveaux d'abstraction



Temps d'encodage puis décodage d'une image MPEG 4

(source *STMicroelectronics*)

Gain en temps de modélisation	
RTL	1
CABA	3
TLM	10

Motivations

Approches tournées vers le développement logiciel

- en avance de phase vis-à-vis disponibilité SoC ou PCB
- participe de la qualité logicielle car facilite les tests de non régression
- facilite le débug grâce à une grande observabilité

Et sa vérification fonctionnelle

En particulier du logiciel lié au matériel

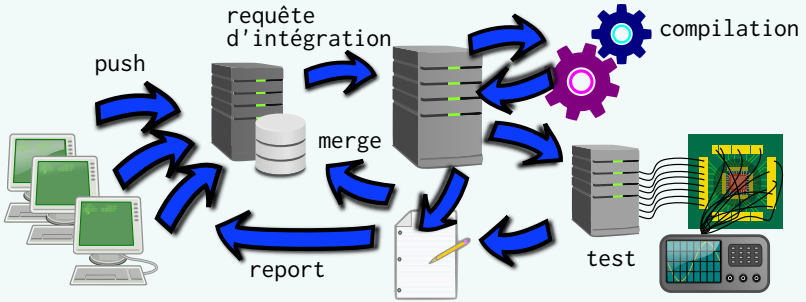
- *firmware*, logiciels *bare metal*
- *drivers* des systèmes d'exploitation

Mais aussi applications parallèles exploitant les plates-formes

Boot de Linux ARM11MPCORE	
hôte	2 s
TLM+DBT	10 s
CABA+ISS	3.3 h

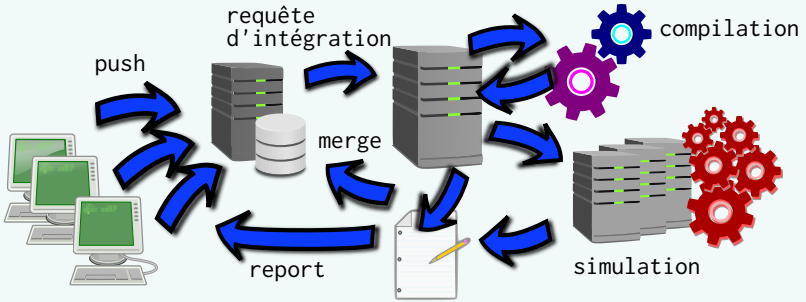
(source TIMA)

Exemple emblématique : intégration continue



Motivations

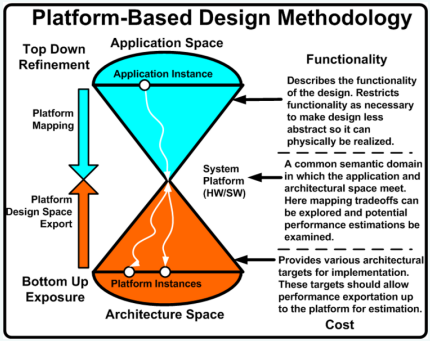
Exemple emblématique : intégration continue



Motivations

Exploration de l'espace de conception matériel et/ou logiciel

- spécification exécutable : sert de référence
- facilité de déploiement et de mise en œuvre
- support aux choix de dimensionnement
  - taille et nombre de bus
  - famille et nombre de processeurs
  - niveau de parallélisations des programmes
- permet l'injection précise de fautes et l'analyse des impacts
- remplace les feuilles excel des experts
- rationalise le processus de décision



(source A. Sangiovanni-Vincentelli, Berkeley)

Propriétés non fonctionnelles

Estimation réaliste délicate

- temps
- puissance consommée
- température

« Truth . . . is much too complicated to allow anything but approximations », John Von Neumann, 1947
« All models are wrong ; some models are useful », George E. P. Box, 2005

Vitesse de simulation

- précision influence grandement la vitesse de simulation
  modélisation des caches, des bus, etc, ralentissent fortement
- vitesse de simulation très critique pour mise au point du logiciel

Plan

Simulation materiel/logiciel

Clarification

hôte : machine sur laquelle la simulation s'exécute

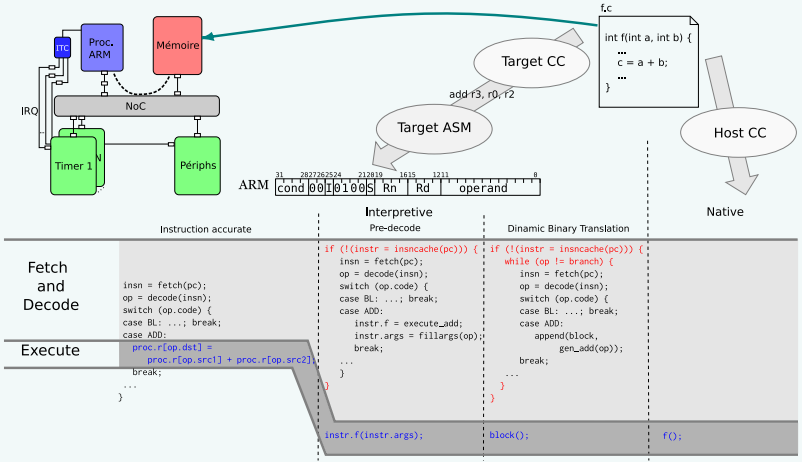
cible : machine simulée

Hypothèses

- simulation repose sur évènements pour exécution du matériel
  - haut niveau d'abstraction pour vitesse compatible avec developpement logiciel
- logiciel à une place à part
  - exécuté par un modèle de processeur

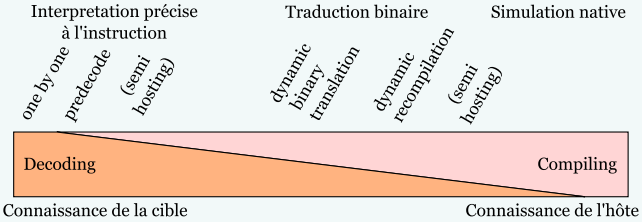
Panel des techniques

Panel des techniques de simulation du logiciel



Panel des techniques

Sur un axe de l'interprétation complète à l'exécution native

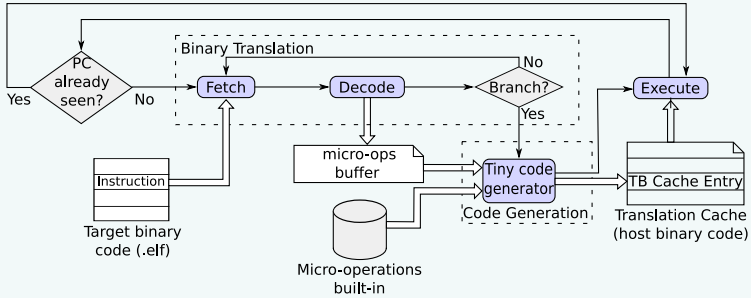


Caractéristiques gros grain des approches de simulation de logiciel

	sim. speed	simulation accuracy		development time		full soft. exec.
		ins. count	time	first	reuse	
IA	−−	++	++	+	+	yes
Predecode	−	++	++	−	+	yes
BT	++	−−	−−	−−	−	yes
Native	+ + +	−−	−−	−−	++	no

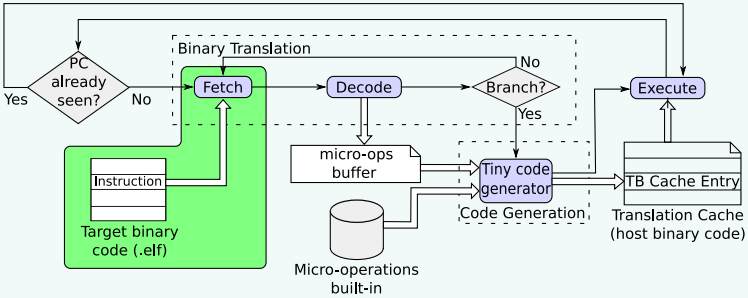
Mécanisme de simulation

Processus d'interprétation



Mécanisme de simulation

Processus d'interprétation

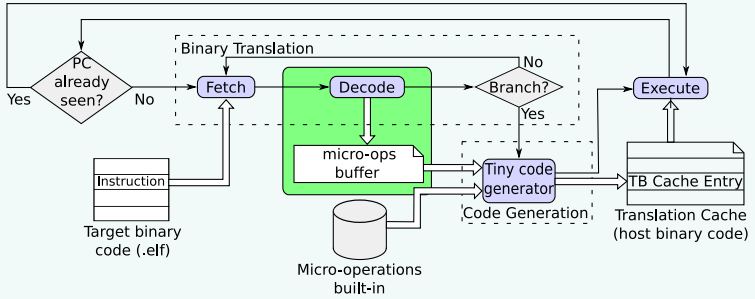


Exemple de génération de code

```
18    target_instrX
```

Mécanisme de simulation

Processus d'interprétation



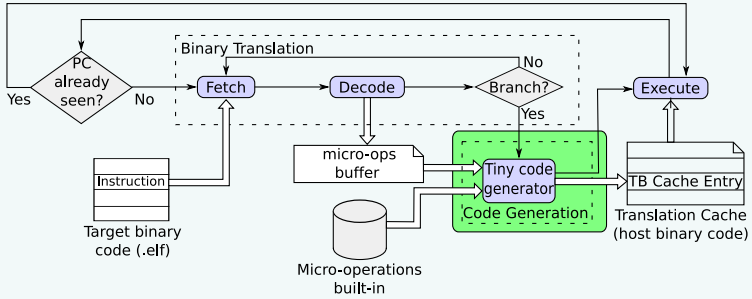
Exemple de génération de code

```
18  target_instrX          micro-op1_instrX

                           micro-op2_instrX
```

Mécanisme de simulation

Processus d'interprétation



Exemple de génération de code

18	target_instrX	micro-op1_instrX	host_instr1_micro-op1_instrX
			host_instr2_micro-op1_instrX
			host_instr3_micro-op1_instrX
		micro-op2_instrX	host_instr1_micro-op2_instrX
			host_instr2_micro-op2_instrX

Mécanisme de simulation

Conséquences

- instructions exécutées par « blocs » en temps zéro
- exécution directe sur hôte avec chaînage des blocs
  impossible au simulateur de reprendre la main si pas prévu

$\Rightarrow$ synchronisation avec les modèles matériels à définir

Points de synchonisation

- opérations d'entrée/sortie (accès aux périphériques)
- gestion des interruptions de l'hôte
- après un nombre d'instructions prédéfini
  par insertion de code *ad-hoc* lors de la génération

Annotation du code généré : principes

Objectifs

Ajouter des synchronisations, quantifier, par ex., le temps d'exécution sur la cible

Insertion de micro-operations

- accumule un nombre de cycles fonction de la spécification
  $\Rightarrow$ prise en compte des dépendances entre registres, données, et contrôle
- modélisation micro-architecture (caches, prédicteurs de branchements, ...)

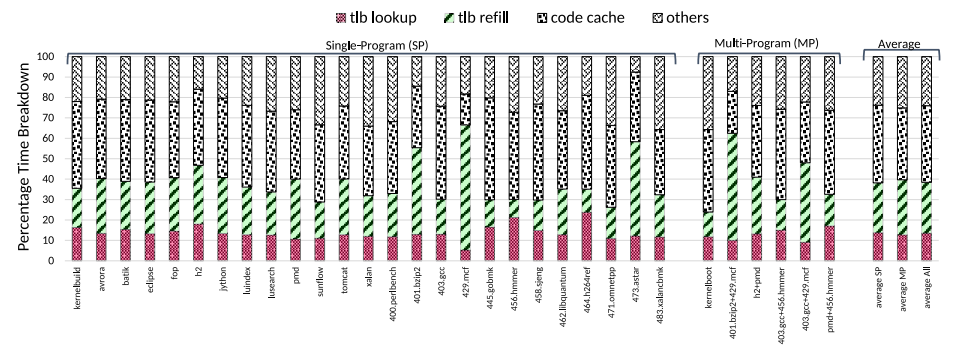
Exemple d'annotations

Instr address	Target code	Original translation	Annotated translation	Annotated generated code
addr_instr1	target_instrX	micro-op1_instrX	micro-op1_instrX	host_instr1_micro-op1_instrX
				host_instr2_micro-op1_instrX
				host_instr3_micro-op1_instrX
		micro-op2_instrX	micro-op_annotation	host_instr1_micro-op_annotation
				host_instr2_micro-op_annotation
			micro-op2_instrX	host_instr1_micro-op2_instrX

Analyse de la performance séquentielle d'un traducteur binaire dynamique

Vitesse séquentielle

Exemple du profilage du temps d'exécution de QEMU



(source X. Tong, T. Koju, and M. Kawahito, IBM Research - Tokyo)

Traduction adresses virtuelles en adresse physiques : 38% temps d'exécution !

Accélération de la traduction d'adresse

Implantation initiale

Virtual TLB

- table de hachage de 4096 ou 8912 entrées avec fonction de *hash* très simple
- collision $\Rightarrow$ remplacement

Premières modifications

- ajout d'une « Victim TLB » de 8 entrées [a]
- collision dans Virtual TLB $\Rightarrow$ copie dans Victim TLB avant remplacement
- échec de consultation dans Virtual TLB $\Rightarrow$ consultation Victim TLB avant *page walk*

$\Rightarrow$ gain de 10% tout de même !

a. Sur le principe énoncé par Jouppi pour les caches dans *Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers*, 17th ISCA, 1990.

Accélération de la traduction d'adresse

Code résultant

Target source code

```
ldr  r3, [r7, #4]
```

Generated host code

Get target virtual address

```
0: movl 0x1c(%r14), %ebp
1: addl $4, %ebp
```

Compute virtual TLB index and tag

```
2: movl %ebp, %edi
3: leal 3(%rbp), %esi
4: shrl $5, %edi
5: andl $0xfffffc00, %esi
6: andl $0x1fe0, %edi
7: leaq 0x2cf0(%r14, %rdi), %rdi
```

Compare tag and call slow path or continue

```
8: cmpl (%rdi), %esi
9: movl %ebp, %esi
10: jne  slow_path_trampoline
```

Fetch data at host virtual address

```
11: addq 0x10(%rdi), %rsi
12: movl (%rsi), %ebp
ret_addr:
```

Slow path trampoline code

```
slow_path_trampoline:
0: mov r14,%rdi
1: mov oi,edx
2: lea -0x7f(%rip),%rcx#ret_addr
3: mov helper_address,%r10
4: callq  *%r10
5: mov %eax,%ebp
6: jmpq ret_addr
```

Prepare architectural state and operation index

Prepare function return address

Call softmmu handler

Retrieve handler return value and return to generated code

Accélération de la traduction d'adresse

Analyse

- chemin court : 13 instructions, dont 6 accès mémoire et 1 branchement
- chemin long : appel à une fonction *ad-hoc*

Peut-on faire mieux ?

- remplacer l'accès mémoire cible par un accès hôte !

Target source code

```
ldr  r3, [r7, #4]
```

Generated host code

```
0: movl 0x1c(%r14), %ebp
1: addl $4, %ebp
2: mov $oi, %rsi
3: mov $MBA, %rdi
4: mov (%ebp,%rdi,1), %rx
```

Get target virtual address

Prepare Handler Arguments

Execute Memory Access

Accélération de la traduction d'adresse

Traductions cible incluses dans traductions hôte

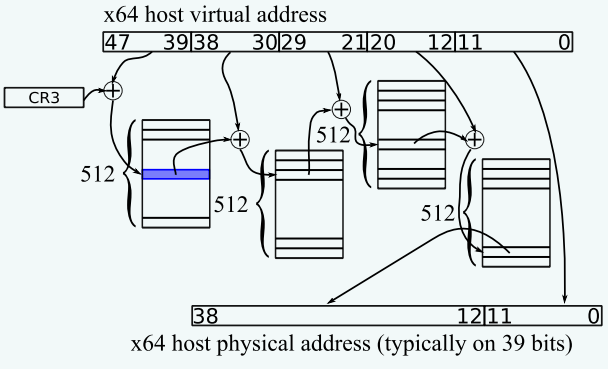
Deux solutions :

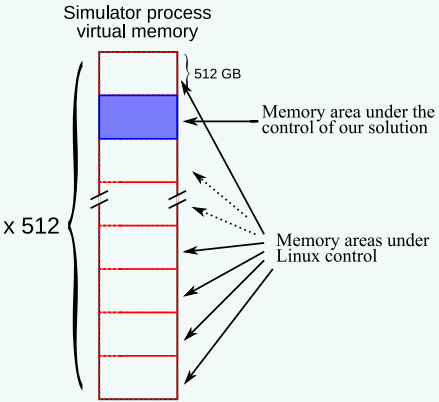
1. exploitation d'un deuxième niveau de tables de pages
   - support matériel à la virtualisation
   - présent sur processeurs modernes
   - permet au simulateur de gérer ses propres pages
   - mais :
     - simulateur superviseur sur processeur virtuel
     - changement de « mode » couteux et fréquents
     - débogue infernal
2. module noyau Linux
   - projection [a] de l'espace d'adressage cible dans une région de l'hôte
   - contrôle de cette projection à travers la MMU hôte

a. J'aurais bien écrit *mapping*, mais Daniel est dans l'auditoire, ...

Accélération de la traduction d'adresse

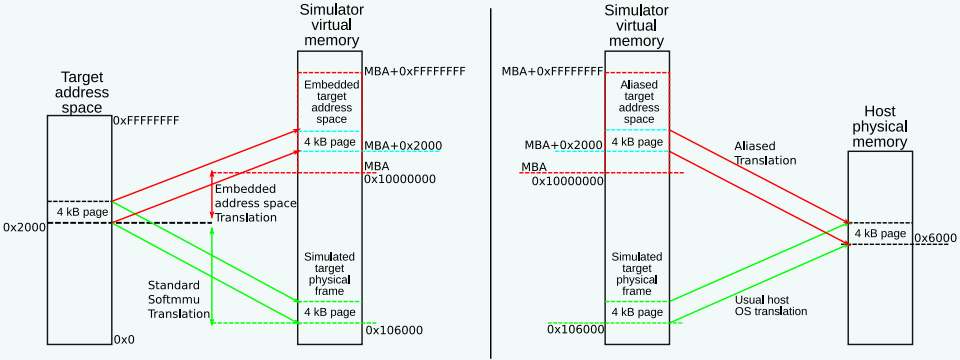
« OPA » hostile sur une partie de la mémoire virtuelle





Accélération de la traduction d'adresse

Projection gérée par `ioctls`



- utilisé uniquement pour code utilisateur, code système utilise traduction classique
- limitation : nb bits adresses virtuelles cible <(<) nb bits adresses virtuelles hôte
- gains de 0% à 300%, croissant avec la durée de la simulation

Autres sources d'accélération séquentielle

Miscellanées

- intégration en ligne, attention à la taille du code
- support *ad-hoc* d'instructions spécifiques : SIMD, 3D, DSP, ...
- spécialisation de function et memoisation [a]
- optimisation du code généré :
  - utilisation d'algorithmes de compilation avancés
  - détection de traces/régions très empruntées (*hot paths*) jouable uniquement si en parallèle de l'exécution [b]

a. Ex. Rohou et al, IRISA, France
b. C.f. par ex. HQemu de Hong et al., Academia Sinica, Tawain

Sources de performance dans la simulation multi-cœurs

Exploitation de la nature SMP de l'hôte :

Parallélisation des CPUs

Traduction des opérations atomiques sur l'hôte [a] :

- par ex. `ll/sc` du Mips II sur `cmpxchg` d'Intel x64

$\Rightarrow$ pas toujours simple, et très architecture dépendent, ...

a. Ex. implantation MTTCG dans QEMU, Konrad et al.

Exécution multitâche des périphériques

- généralement implantée au dessus des *pthread* :
  - avec des synchros type *read-copy-update* dans Qemu
  - avec les objets des *pthread* dans SystemC

Sources de performance dans la simulation

Exploitation de la nature SMP de l'hôte :

Parallélisation du simulateur

Vieux serpent de mer, ...

- simulation à événements discrets parallèle, ... *back to the future* [a]
- OoO PDES :
  - pas de modification des modèles, mais recompilation :
    - construction de « segments » : d'un *wait* au suivant, conservativement
    - exécution parallèle des segments des différents modèles
    - $\Rightarrow$ accroissement désynchronisation utilisant informations temporelles des modèles
    - $\Rightarrow$ nombre de segments indépendants potentiellement très élevé [b]
    - $\Rightarrow$ accélération linéaire ... dans cas idéal

a. Chandy/Misra/Bryant, Fujimoto, ...
b. Bernard aimerait exécuter un code de ce genre, non ?

Conclusion(s)

Méthodes de « simulation » de logiciel

Traduction binaire dynamique

- ⇒ IMHO la « bonne » technologie :
  exécution du logiciel cross-compilé sans modification
  rapidité raisonnable, $\frac{1}{5}$ à $\frac{1}{50}$ de la vitesse native
  reciblable, solution open-sources disponibles, ...
- ⇒ bénéficie de décennies de recherche et développement :
  Smalltalk, p-code, Java, ...
- ⇒ gains potentiels à la marge
- ⇒ mais encore d'actualité :
  plusieurs papiers dans CGO, TACO, DATE, etc, ces dernières années

Conclusion(s)

Simulation des multicœurs

Passe par la parallélisation

$\Rightarrow$ très compliqué d'être innovant et général

Conclusion(s)

A propos du prototypage virtuel au niveau système

Indépendamment de la stratégie de simulation du logiciel

Intégration avec l'écosystème pas triviale

- acceptation encore difficile en dehors de niches
- qualité des modèles
- fonction seule ne suffit pas :
  « trop lent », « pas précis », « pas assez précis »
- rôle important de la standardisation :
  SystemC+IPXact dans monde SoC