Adaptation de micro-architectures hétérogènes via l'utilisation de gem5 et CERE

Nicolas Derumigny Pablo de Oliveira Castro

ENS Lyon UVSQ

5 Septembre 2016

- Utilisation de morceaux de codes via CERE[2]
- Simulation de différents CPU via gem5[1]
- Simulation de la consommation énergétique via McPAT[3]
- Calcul du ratio performance-consommation énergétique

Codelet Extractor and REplayer : Définitions

CERE extrait des morceaux de code d'une application, créant ainsi une autre application autonome, reproduisant au plus près possible son exécution, autant dans ses instructions que dans son contexte.

Codelet

- Portion d'un application.
- *In-vivo* : fonctionnement naturel à l'intérieur de son application.
- *In-vitro* : fonctionnement dans l'application produite par CERE.

CERE : Un outil d'isolation de morceaux de code

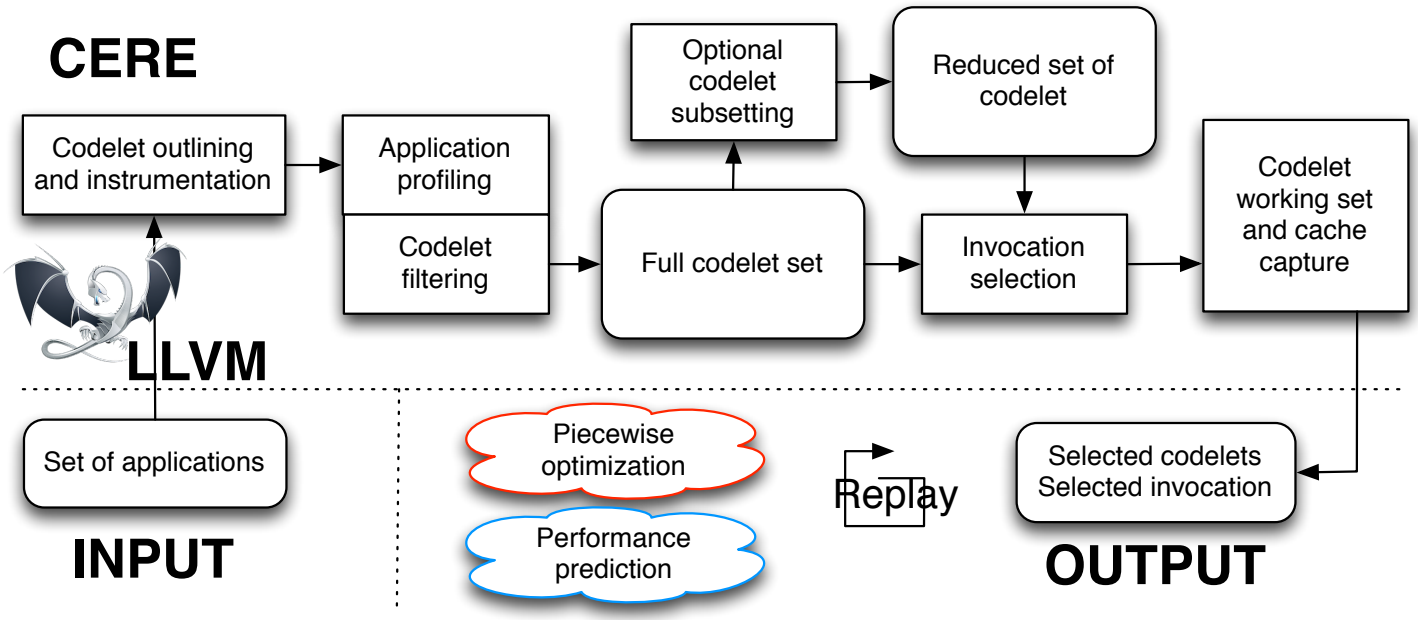

Figure – Schéma fonctionnel de CERE.

Gem5 : Un simulateur précis au cycle près

- Projet open-source, codé en C++
- Utilisation de python pour le choix des configurations
- Support des jeux d'instruction ARM, x86, Alpha, SPARC, POWER et MIPS
- Simulation du fonctionnement du CPU cycle par cycle
- Simulation de systèmes hétérogènes
- Accès aux données non mesurables d'un CPU standard (cache miss)

Gem5 : Le mode *syscall emulation* (SE)

- Émulation d'une application au sein d'un système linux complet virtuel
- Besoin de codage séparé de tous les appels systèmes
- Pas de support des libraires dynamiques
- Systèmes multi-cœurs peu flexible (m5thread)
- Exécution d'une seule application

Gem5 : Le mode *fullsystem* (FS)

- Émulation d'un CPU "nu"
- Exécution d'un système linux complet par le CPU simulé
- Support des libraires dynamiques et des systèmes multi-cœurs via le noyau linux
- Exécution d'une suite quelconque de commandes
- Plus lent que le mode SE
- Nécessite au préalable la préparation d'une image disque

McPAT : Un simulateur de conception et de consommation de puces

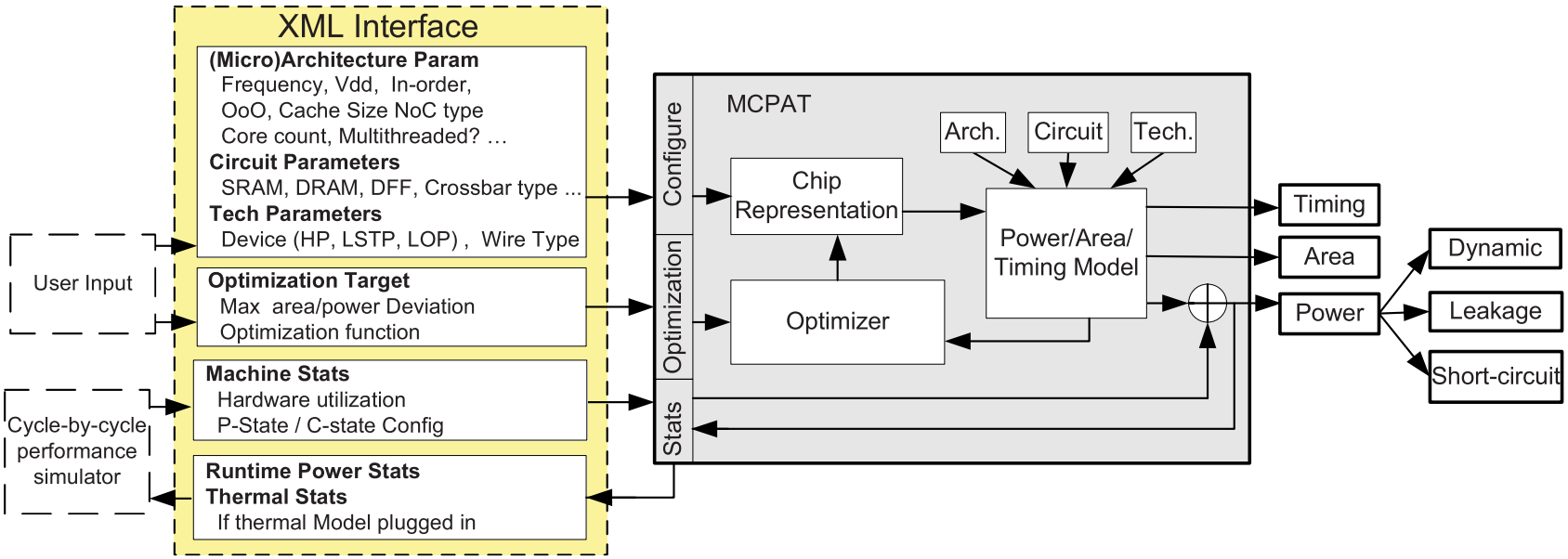

Figure – Schema fonctionnel de McPAT.

Correction de *readFunc()*

```
1  SyscallReturn
2  readFunc(SyscallDesc *desc, int num, LiveProcess *p, ThreadContext *tc)
3  {
4      int index = 0;
5      int tgt_fd = p->getSyscallArg(tc, index);
6      Addr bufPtr = p->getSyscallArg(tc, index);
7      int nbytes = p->getSyscallArg(tc, index);
8      BufferArg bufArg(bufPtr, nbytes);
9
10     int sim_fd = p->getSimFD(tgt_fd);
11     if (sim_fd < 0)
12         return -EBADF;
13
14     int bytes_read = read(sim_fd, bufArg.bufferPtr(), nbytes);
15
16     if (bytes_read > 0)
17         bufArg.copyOut(tc->getMemProxy());
18
19     return bytes_read;
20 }
```

Ajout des appels système *getdents()* et *getdents64()*

```
1  SyscallReturn
2  getdentsFunc(SyscallDesc *desc, int num, LiveProcess *p, ThreadContext *
       tc)
3  {
4      int index = 0;
5      int sim_fd = p->sim_fd(p->getSyscallArg(tc, index));
6      if (sim_fd < 0)
7          return -EBADF;
8      Addr buf_ptr = p->getSyscallArg(tc, index);
9      int nbytes = p->getSyscallArg(tc, index);
10     BufferArg buf_arg(buf_ptr, nbytes);
11
12     int nread = syscall(SYS_getdents, sim_fd, buf_arg.bufferPtr(),
       nbytes);
13     int errno_after_call=errno;
14
15     if (nread > 0)
16         buf_arg.copyOut(tc->getMemProxy());
17
18     return (nread == -1) ? -errno_after_call : nread;
19 }
```

Simulation de quatre processeurs aux caractéristiques diverses

Nom	Fréquence	Assoc.	L2		L3
			Taille	Assoc.	
A15	1 GHz	2	1 Mo	16	Non
i5-3550	3,3-3,7 GHz	8	256 ko	8	8 Mo / 16
i5-3337U	1,8 GHz	8	256 ko	16	4 Mo / 8
Q9100	2,26 GHz	8	8 Mo	16	Non

Figure – Paramètres utilisés pour chaque CPU. Le reste de la configuration est fixe : 8 Go de RAM, processeur x86 générique gravé en 44nm, et un unique cœur simulé.

Trois codelets sur le banc d'essai

- NAS IS séquentiel : région __cere__is_ranked_475, qui vérifie si un tableau est bien trié.
- PARSEC blackscholes : région __cere__blackscholes_m4__Z9bs_threadPv_first, une région parallèle OpenMP calculant la valeur d'une option en se basant sur l'équation de Black & Scholes.
- PARSEC freqmine : région __cere__tree8scan1_DBEP4Data_first, générant un hash à partir d'un arbre de données.

IS : Une première application simple

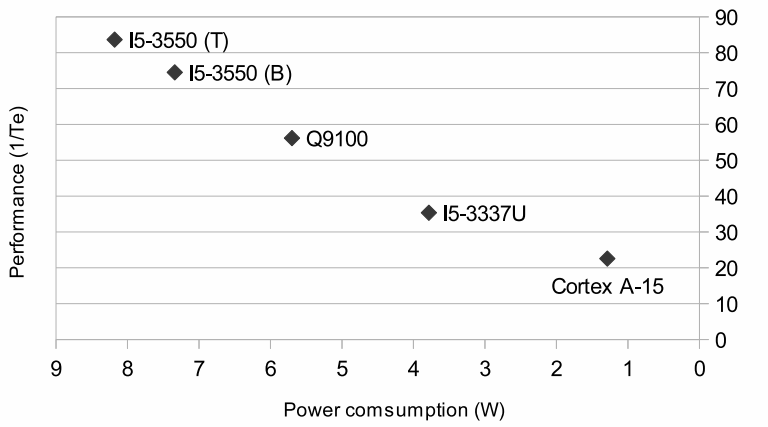

Figure – Graphique du ratio performance-consommation énergétique du codelet *IS*.

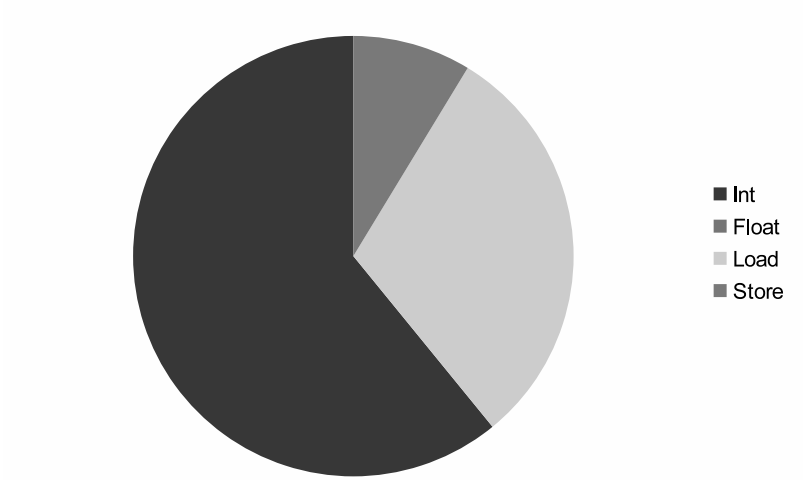

Figure – Répartition des instructions au cours de l'exécution du codelet *IS*.

Freqmine : Une application parallèle à peu de données

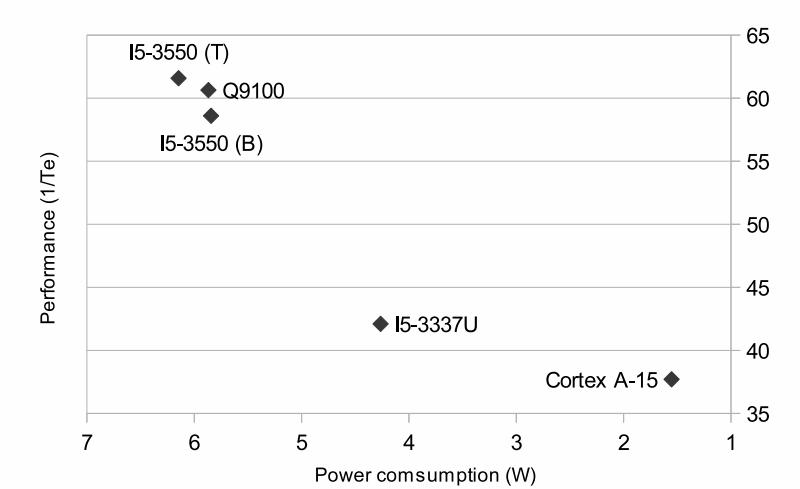

Figure – Graphique du ratio performance-consommation énergétique du codelet *Freqmine*.

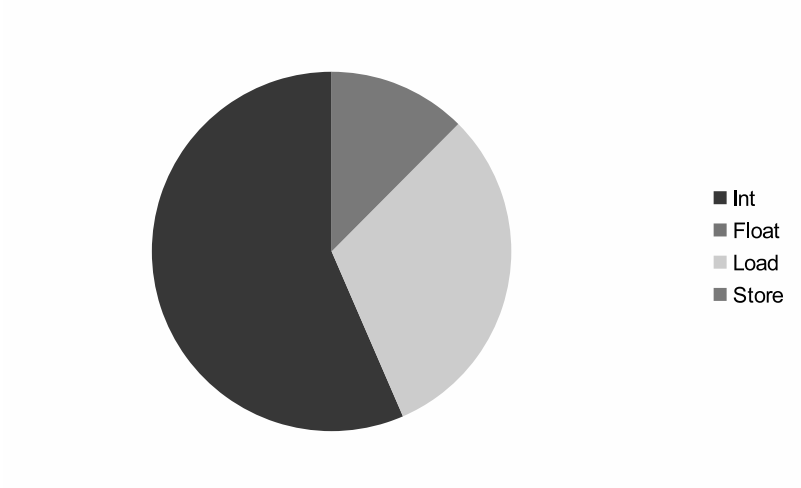

Figure – Répartition des instructions au cours de l'exécution du codelet *Freqmine*.

Blackscholes : Une application utilisant significativement les flottants

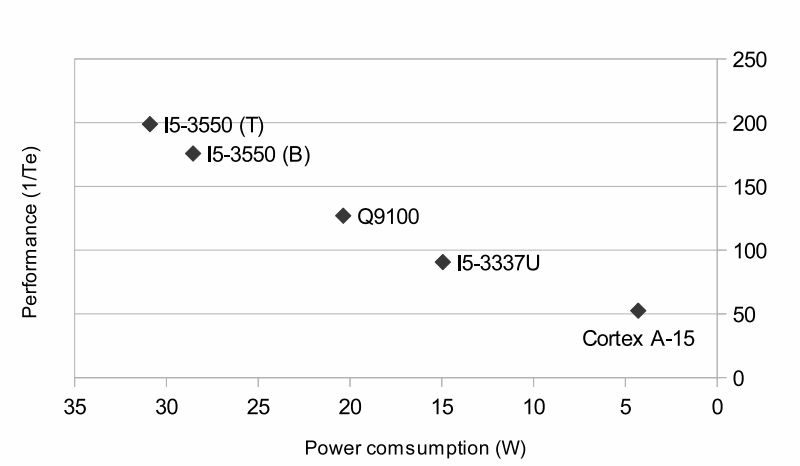

Figure – Graphique du ratio performance-consommation du codelet *Blackscholes*.

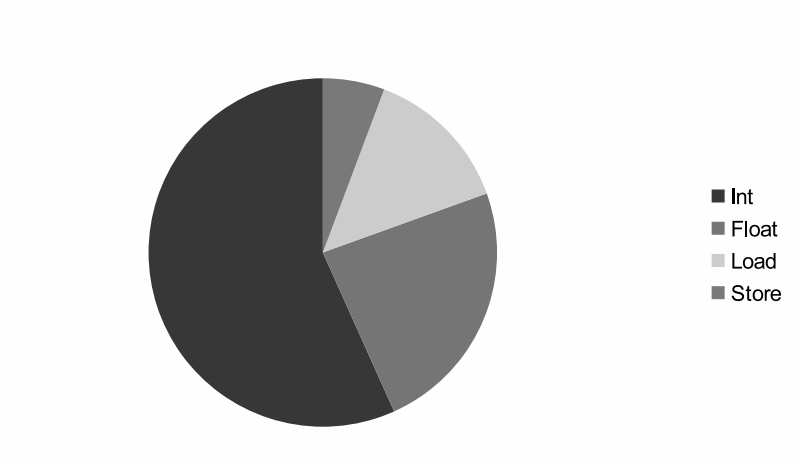

Figure – Répartition des instructions au cours de l'exécution du codelet *Blackscholes*.

Définition du ratio performance-consommation énergétique

Ratio performance-consommation énergétique

Le ratio performance-consommation énergétique I est défini par :

$$I = \frac{1}{P.t_e}$$

Avec P la puissance (en W) du processeur simulé et T_e le temps d'exécution du codelet (en s).

Note : Cela correspond à l'inverse de la consommation totale du processeur.

Graphe récapitulatif

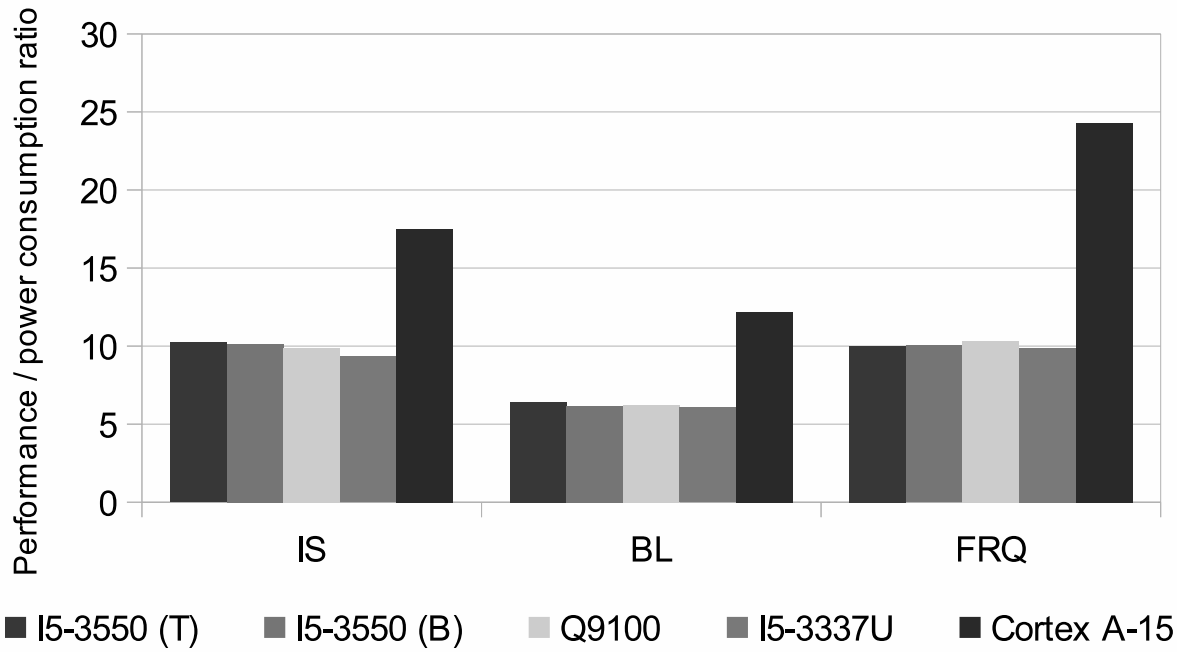

Figure – Ratio performance-consommation énergétique pour chaque codelet.

- Des résultats cohérents
- Des performances plutôt indépendantes du jeu d'instruction
- Influence importante de la taille du cache, parfois plus que la puissance de calcul brute
- Gain de temps induit par l'utilisation de codelets
- Utilisation possible comme étalon d'un ordonnanceur sur une plateforme hétérogène

Bibliographie

Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood.
The gem5 simulator.
SIGARCH Computer Architecture News, 39(2) :1–7, 2011.

Pablo de Oliveira Castro, Chadi Akel, Eric Petit, Mihail Popov, and William Jalby.
CERE : llvm-based codelet extractor and replayer for piecewise benchmarking and optimization.
TACO, 12(1) :6, 2015.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi.
Mcpat : an integrated power, area, and timing modeling framework for multicore and manycore architectures.
In David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez, editors, *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, pages 469–480. ACM, 2009.

Compilation croisée depuis l'architecture x86 vers Aaarch64

- Installation du cross-compilateur LLVM
- Modification de CERE : compilateur linker et objdump utilisé
- Changement de l'adresse de départ de l'exécutable
- Changement d'adresse de la pile (cf figure)

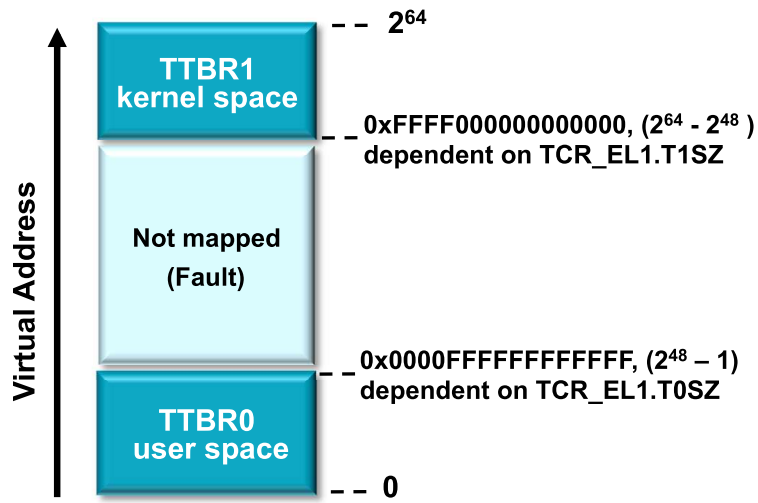

Figure – Mappage de la mémoire virtuelle de l'architecture aarch64