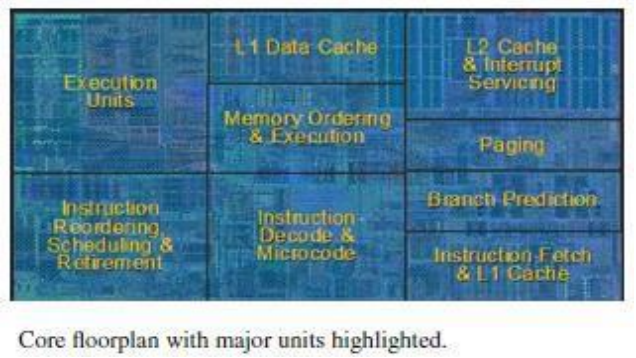
TD/TP4 : Microprocesseur superscalaire/mémoires caches - Analyse de configurations d'architectures de microprocesseurs

Exercice 1 : Prédiction de branchement

La prédiction de branchement permet aux processeurs d'exécuter un flot d'instructions sans pénalité de blocage dû à l'attente de l'évaluation de la condition de branchement. Tous les microprocesseurs haute-performance incluent une unité de prédiction de branchement. La Figure 6 montre la surface prise par l'unité de prédiction de branchement dans un processeur multi-cœur Intel.



Core floorplan with major units highlighted.

Figure 6 Floorplan d'un cœur de processeur Intel

Nous souhaitons dans cet exercice évaluer l'impact de différents mécanismes de prédiction de branchement sur le CPI et le nombre de cycles pour 2 applications :

1. La fonction de hachage SHA-1 ([7] [9]). Voir l'annexe « Applications ».
2. L'algorithme de calcul de la centralité d'intermédiarité (Betweenness Centrality Score / BCS) dans la suite SSCA2 [TP2.1], se référer à l'annexe « Applications ».

Les mécanismes de prédiction que nous souhaitons comparer sont les suivants :

- **nottaken :** ce mécanisme de prédiction de branchement statique considère toujours que le branchement n'est pas pris
- **taken :** ce mécanisme de prédiction de branchement statique considère toujours que le branchement est pris
- **perfect :** ce prédicteur « virtuel » représente la prédiction parfaite
- **bimod :** le prédicteur de branchement bimodal utilise un BTB (Branch Target Buffer) avec compteurs à 2 bits
- **2lev :** prédicteur de branchement adaptatif à 2-niveaux

Vous pouvez spécifier en utilisant le simulateur SimpleScalar le mécanisme de prédiction de branchement de votre choix en utilisant la commande suivante :

sim-outorder –bpred <type> <prog>.ss [options]

La documentation de SimpleScalar (voir l'annexe en fin de document) précise les détails pour les différents mécanismes de prédiction de branchement. La Figure 7 et la Figure 8 décrivent 2 types de prédicteurs de branchement, et la Figure 9 décrit les différents paramètres en entrée pour le prédicteur de branchement de SimpleScalar.

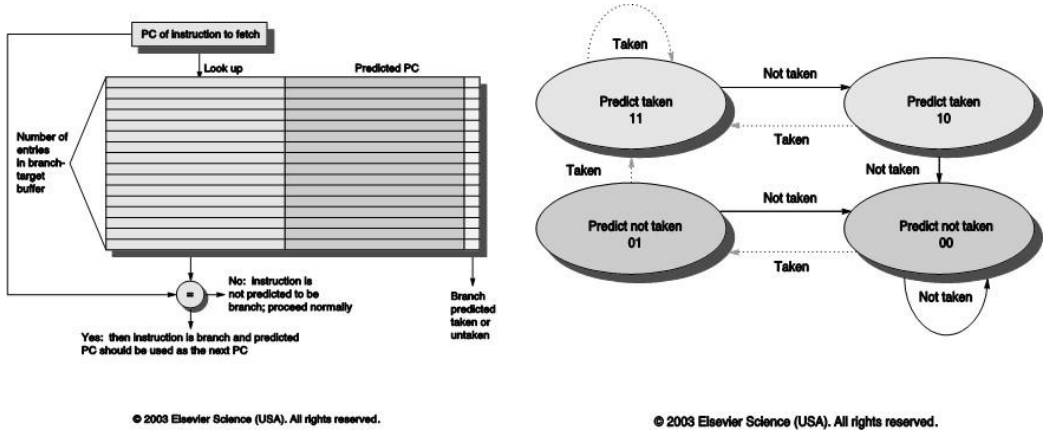


Figure 7 Le prédicteur de branchement bimodal

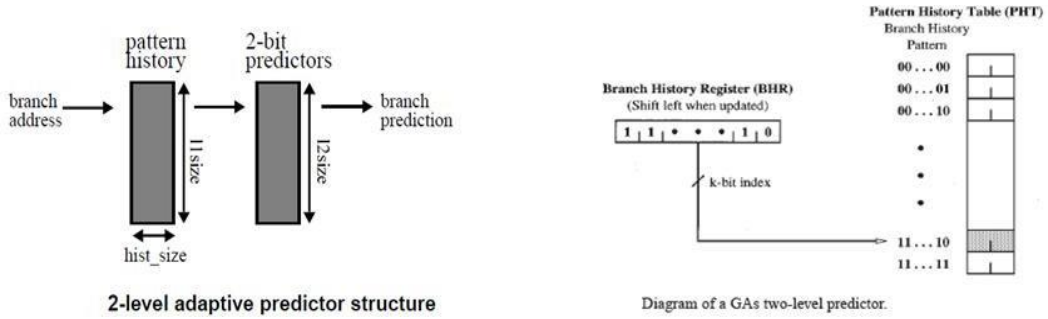


Figure 8 Le prédicteur de branchement adaptatif à 2-niveaux

predictor	l1_size	hist_size	l2_size	xor
GAg	1	W	2^W	0
GAp	1	W	$>2^W$	0
PAg	N	W	2^W	0
PAp	N	W	2^{N+W}	0
gshare	1	W	2^W	1

Branch predictor parameters

Figure 9 Paramètres du prédicteur de branchement

Q1 : *Etudier l'impact des différents prédicteurs de branchement sur le CPI et le nombre de Cycles pour le programme de votre choix*

Exercice 2 : Impact de la fenêtre d'instructions RUU

Le modèle d'exécution superscalaire de SimpleScalar utilise un mécanisme de ré-ordonnancement basé sur une partie centrale : la RUU (Register Update Unit).

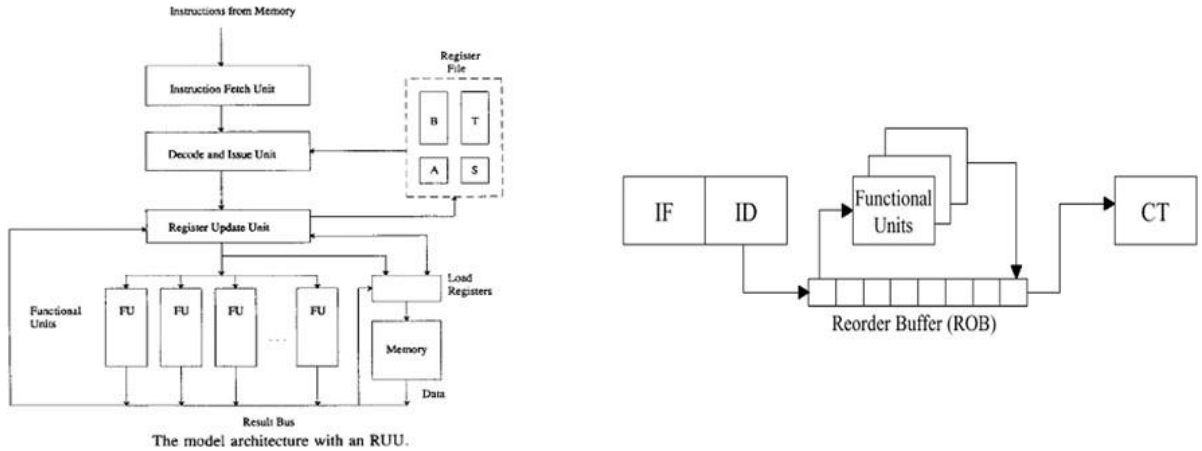


Figure 10 Register Update Unit (RUU)

L'augmentation de la taille de la RUU permet à priori de considérer davantage d'instructions potentielles pour l'exécution. Nous souhaitons analyser l'impact de cette taille sur le CPI.

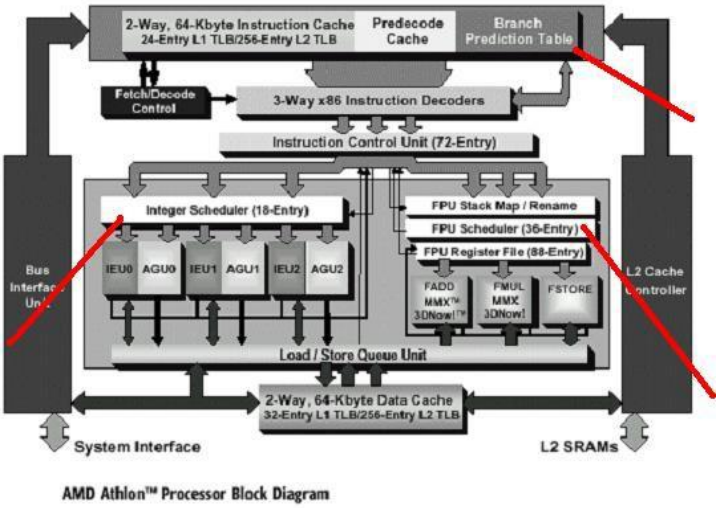


Figure 11 AMD Athlon Processor Block Diagram

Q1 : *Faites varier la taille de la RUU de 16 à 128 et évaluez l'impact sur le CPI et le nombre de cycles sur une application de votre choix (SHA-1, SSCA2,…).*

Exercice 3 : Mémoires caches - Evaluation des performances de différentes configurations de mémoires caches (instructions et données) pour 4 algorithmes de multiplication de matrices.

De nombreuses applications informatiques dans différents domaines font appel à des opérations sur des matrices, et en particulier la multiplication de matrices. Cette opération classique peut être répétée de nombreuses fois au cours de l'exécution de ces applications et il est donc primordial que ses performances soient optimisées. La multiplication de matrices sur des matrices de dimensions importantes est une opération dont la performance en temps d'exécution dépend beaucoup des performances de la hiérarchie mémoire, et notamment l'organisation des caches du microprocesseur sur lequel elle s'exécute. Le taux de défauts de caches **(miss rate),** qui est le rapport entre le nombre d'accès à un cache qui se traduisent par un défaut de cache **(cache miss)** sur le nombre total d'accès au cache, est le paramètre principal pour évaluer cette performance. Nous considérons ici des microprocesseurs ayant des caches d'instructions et de données séparés. De ce fait, le miss rate du cache d'instructions et du cache de données méritent une analyse.

Un simulateur de cache, comme **sim-cache** (**Annexe 1**), permet d'effectuer une évaluation du miss rate pour une configuration de caches particulière et pour un programme en entrée. Lorsque l'on souhaite évaluer plusieurs configurations de caches il est alors nécessaire d'effectuer autant de simulations. Ce processus se répète pour le nombre de programmes que l'on considère en entrée.

Travail demandé

Nous considérons les 2 organisations de caches décrites dans le Tableau 7. On supposera que l'algorithme de remplacement est LRU pour toutes les configurations.

Tableau 7 Configurations de caches pour 2 processeurs

Configuration	Instruction cache	Data cache	L2 cache	Block size (bytes)
C1	4KB direct-mapped	4KB direct-mapped	32KB direct-mapped	32
C2	4KB direct-mapped	4KB 2-way set-asso	32KB 4-way set-asso	32

Les binaires pour les 4 algorithmes de matrices se trouve dans :
/usr/ensta/pack/simplescalar-2.0/simplescalar-4.0/mase/matrix/<algo>.ss

Q1 : *Pour chacune des 2 configurations de mémoires cache, complétez le Tableau 8. Pour cela, vous devez déterminer les paramètres d'entrée du simulateur de cache « sim_cache » de SimpleScalar.*

Tableau 8 Paramètres de sim-cache pour chaque configuration

Configuration	IL1	DL1	UL2
C1			
C2			

Q2 *: Complétez les tableaux 9, 10 et 11. Pour cela vous devez simuler à l'aide de sim-cache les différentes configurations et collecter les informations suivantes :*

- Le taux de défauts dans le cache d'instructions il1 : **il1 . miss_rate**
- Le taux de défauts dans le cache de données dl1 : **dl1.miss_rate**
- Le taux de défauts dans le cache unifié (L2) ul2 : **ul2.miss_rate**

Tableau 9 Instruction Cache (il1) Miss Rate

Programmes	***Configurations de caches***	
	C1	C2
P1 (normale)		
P2 (pointeur)		
P3 (tempo)		
P4 (unrol)		

Tableau 10 Data Cache (dl1) Miss Rate

Programmes	***Configurations de caches***	
	C1	C2
P1 (normale)		
P2 (pointeur)		
P3 (tempo)		

P4 (unrol)		

Tableau 11 Unified Cache (ul2) Miss Rate

Programmes	***Configurations de caches***	
	C1	C2
P1 (normale)		
P2 (pointeur)		
P3 (tempo)		
P4 (unrol)		

Q3 *: Les 4 algorithmes de multiplication de matrices présentent-ils une bonne localité de références pour le code ? Pourquoi ?*

Exercice 4 : Mémoires caches - Evaluation des performances de différentes configurations de mémoires caches (instructions et données)

Description du problème

Un panorama détaillé de l'offre actuelle des composants microprocesseurs (8 bits, 16 bits, 32 bits et 64 bits) peut être consulté sur le site EDN (EDN 2014 Microprocessor Directory [1]). Il existe plus de 70 fabricants de microprocesseurs et de microcontrôleurs, chacun se positionnant soit dans le domaine généraliste soit dans un domaine spécifique. Par exemple, le marché des processeurs généralistes pour serveurs et desktops est dominé par Intel [2] et AMD [3], tandis que le marché des processeurs embarqués est dominé par ARM [4] et MIPS [5]. Au sein des processeurs embarqués, on trouve généralement une classification supplémentaire selon le domaine applicatif (automobile, image/vidéo, militaire/espace, télécommunications, etc.).

Cependant, chez les fabricants de ces deux grandes familles de processeurs (généralistes et embarqués), la tendance commune est à l'élargissement de la gamme de processeurs disponibles : processeurs très performants mais énergivores d'un côté, processeurs légèrement moins performants mais plus efficaces de l'autre. Cela rend les frontières entre ces deux mondes de plus en plus floues, mais cela dégage un objectif commun à tous les fabricants : la recherche de l'efficacité énergétique (ratio performances pures à consommation d'énergie donnée), en témoigne la nouvelle architecture **big.LITTLE** de **ARM** [6].

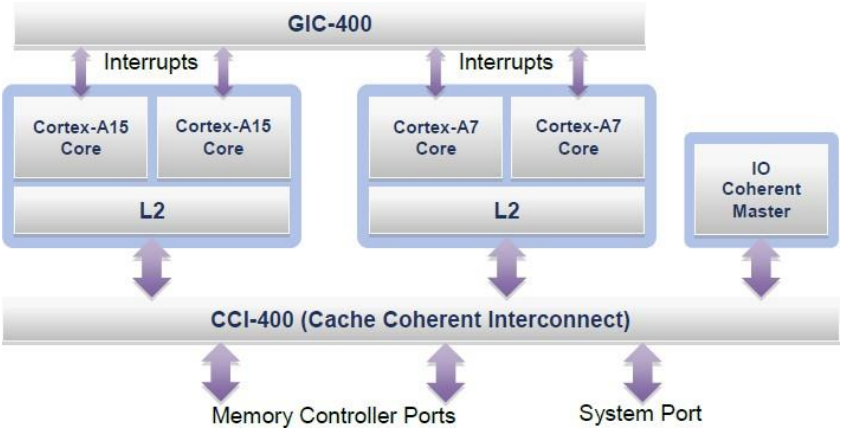


Figure 12 Une configuration typique de big.LITTLE, constituée de 2 clusters, avec 2 CPUs par cluster (Source : ARM)

Cette architecture intègre au sein du même processeur deux cœurs ARM embarqués : un **Cortex A15** (hautes performances, [4]) et un **Cortex A7** (haute efficacité énergétique, [4]). Ces deux cœurs étant entièrement compatibles au niveau du jeu d'instructions, l'architecture peut décider de reloger l'exécution sur l'un ou l'autre des cœurs en fonction des besoins en calcul de l'application. Ces deux cœurs fonctionnent donc de manière alternative (un cœur ON - un cœur OFF), afin de maximiser l'efficacité de l'architecture globale.

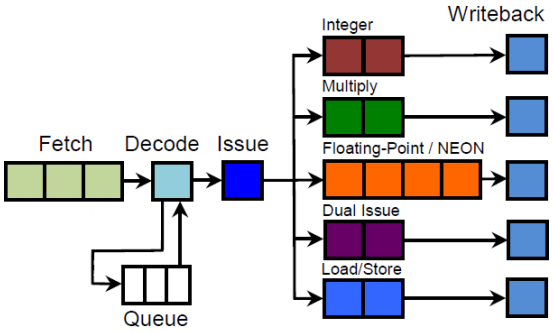


Figure 13 Pipeline du Cortex A7

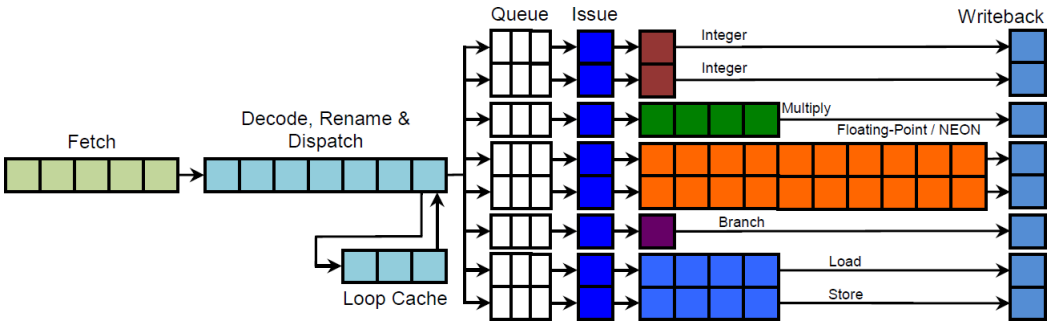


Figure 14 Pipeline du Cortex A15

Comme on a pu le voir, les critères exigés par chaque domaine applicatif (performances, énergie, surface) influencent les choix architecturaux. La microarchitecture d'un processeur spécifie : (1) le mode de traitement des instructions (pipeline, parallélisme d'instructions), (2) l'exécution dans l'ordre ou dans le désordre et enfin (3) le nombre et le type d'unités fonctionnelles (additionneurs, multiplieurs, diviseurs, unités spécialisées, etc.). Il est possible d'identifier pour un programme donné la classe (le type) des instructions sollicitées et le pourcentage d'instructions exécutées issues de cette classe. Cette phase d'identification s'appelle le **profiling**, et a déjà été abordée dans les TD/TP précédents. Il est clair que si le profiling d'une application montre qu'il existe un pourcentage élevé d'utilisation d'une classe particulière, il est alors souhaitable, pour accroître les performances, d'augmenter le nombre d'unités fonctionnelles correspondant à ce type. A contrario, les classes d'instructions ayant un faible taux d'utilisation pourraient voir leur nombre d'unités associées réduites, voire éliminées.

Ce processus consiste à faire une **exploration manuelle/automatique** de la microarchitecture en modifiant les paramètres de configuration de l'architecture et à évaluer l'impact sur les performances et la surface. Néanmoins, ce processus s'applique plutôt à des processeurs en cours de conception. Dans le cas d'une offre existante, il s'agit plutôt d'estimer les performances potentielles en modélisant des processeurs existants.

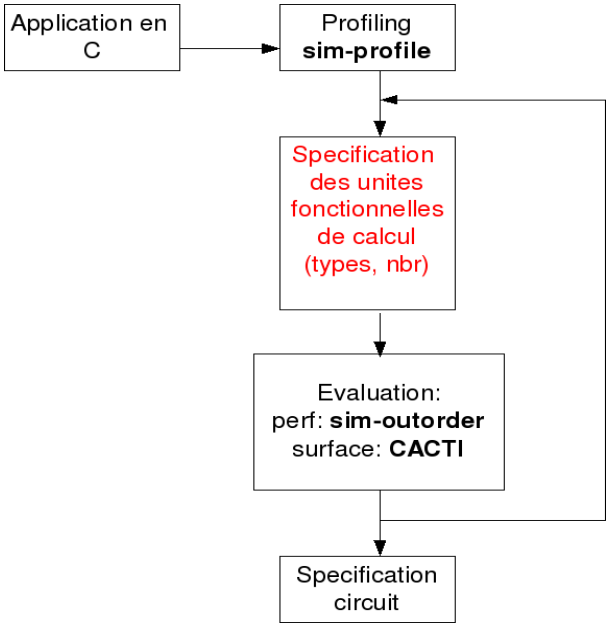


Figure 15 Flot de modification de la microarchitecture d'un microprocesseur

Le but de ce TD/TP est d'analyser les performances des 2 cœurs ARM décrits en introduction : **Cortex A7** et **Cortex A15**, sur des applications type traitement de graphes. Les configurations de ces cœurs ainsi qu'une configuration classique de leurs architectures de caches sont décrites dans le tableau suivant

Tableau 12 Paramètres de configuration des processeurs

Cœur	I-L1$ (cache/ bloc/ assoc.)	D-L1$ (cache/ bloc/ assoc.)	L2$ (cache/ bloc/ assoc.)	Prédicteur de branchement		Fetch queue	Decode/ Issue/ Commit	RUU/ LSQ	# ALU entiers/ flottants	# multip. entiers/ flottants
				Type	Latence mis-préd. (cycles)					
Cortex A15	32KB/ 64/ 2	32KB/ 64/ 2	512KB/ 64/ 16	2 level BTB=256	15	8	4/8/4	16/16	5/1	1/1
Cortex A7	32KB/ 32/ 2	32KB/ 32/ 2	512KB/ 32/ 8	bimodal BTB=256	8	4	2/4/2	2/8	1/1	1/1

Pour les applications, la suite de benchmarks pour l'embarqué **MiBench** sera utilisée et notamment l'applicatif Dijkstra qui recherche les plus court chemins d'un graphe. Cette suite est téléchargeable sur le site http://www.eecs.umich.edu/mibench/ (Téléchargement du code source : wget http://www.eecs.umich.edu/mibench/network.tar.gz). Une description du benchmark est disponible ici : http://www.eecs.umich.edu/mibench/publications/Mibench.pdf

Pour compléter, l'étude sera également menée sur le bloc cipher BlowFish [9] [10], voir l'Annexe 2 « Applications ».

- Compilez l'application **Dijkstra** et **BlowFish** avec **sslittle-na-sstrix-gcc** en modifiant au besoin les *Makefile* associés.

Questions

1. **Profiling de l'application :**

Effectuez un profiling de l'application **dijkstra et BlowFish** en utilisant le simulateur **sim- profile (dijkstra_small *input.dat et* bf.ss *input_small.asc).***

Q1 : Générez le pourcentage de chaque classe d'instructions de ces applications et remplissez les valeurs dans un tableau.

Q2 : Quelle catégorie d'instructions nécessiterait une amélioration de performances ? Expliquez en quelques lignes (max 5 lignes).

Q3 : Au regard des résultats obtenus lors du TP2, pouvez-vous justifier d'éventuelles similitudes/divergences comportementales entre dijkstra, BlowFish, SSCA2-BCS, SHA-1 et le produit de polynômes ?

2. **Evaluation des performances :**

- Cortex A7 : A taille de cache L2 fixe (512KB) et en faisant varier simultanément la taille des caches L1 d'instructions et de données de 1KB à 16KB (i.e. : 1KB, 2KB, 4KB, 8KB, 16KB), effectuez une évaluation de performances en utilisant le simulateur **sim-outorder**.

Q4 : Générez les figures de performances détaillées (performance générale, IPC, hiérarchie mémoire, prédiction de branchement, etc.) en fonction de la taille du cache L1 pour les configurations testées. Analysez les résultats. Quelle configuration de L1 donne les meilleures performances pour le Cortex A7 pour dijkstra ? et pour BlowFish ?

N.B. : *Mentionnez les paramètres d'exécution de sim-outorder que vous avez utilisés.*

- Cortex A15 : A taille de cache L2 fixe (512KB) et en faisant varier simultanément la taille des caches L1 d'instructions et de données de 2KB à 32KB (i.e. : 2KB, 4KB, 8KB, 16KB, 32KB), effectuez une évaluation de performances en utilisant le simulateur **sim-outorder**.

Q5 : Générez les figures de performances détaillées (performance générale, IPC, hiérarchie mémoire, prédiction de branchement, etc.) en fonction de la taille du cache L1 pour les configurations testées. Analysez les résultats. Quelle configuration de L1 donne les meilleures performances pour le Cortex A15 pour dijsktra ? et pour BlowFish ?

N.B. : *Mentionnez les paramètres d'exécution de sim-outorder que vous avez utilisé.*

3. Efficacité surfacique :

La surface du Cortex A15 de l'énoncé (Tableau 13) et de ses caches L1 est de **2 mm²** pour une technologie de **28 nm**. De même, la surface du Cortex A7 de l'énoncé et de ses caches L1 est de **0.45 mm²** dans cette même technologie de **28 nm**. Utilisez l'outil CACTI de HP avec la technologie convenable pour estimer la surface des caches pour chaque type de processeur. Il est recommandé de télécharger la dernière version de CACTI (CACTI 6.5 : http://www.hpl.hp.com/research/cacti/cacti65.tgz) sur vos machines, car il s'agit de la version la plus récente, et génère donc les chiffres d'estimation de caches les plus précis. L'outil CACTI est exécuté avec la commande suivante :

./cacti -infile cache.cfg

N.B. : *Si votre version ne supporte pas la technologie 28nm, ciblez la technologie 32nm.*

Q6 : Observez le fichier de configuration de cache « cache.cfg ». Quels sont les paramètres de cache (taille de cache, de bloc, associativité) et la technologie en **nm** utilisés par défaut ?

Q7 : Quelle est la surface des caches L1 du Tableau 14 (instructions et données) en **mm²** ? Quel pourcentage de la surface totale des cœurs A7 et A15 est occupé par les caches L1 ? En déduire la taille des deux cœurs (hors caches L1). Donnez votre analyse.

Q8 : Faire varier la taille des caches L1 dans l'intervalle de valeurs possibles pour le Cortex A7 et le Cortex A15, et donner (en **mm²**) les surfaces des caches L1 obtenues. Avec la configuration de cache L2 utilisée précédemment (512KB), en déduire pour chaque valeur les nouvelles surfaces totales du Cortex A7 et du Cortex A15, caches L2 inclus (présentez les résultats sous forme de graphes).

Q9 : En prenant en compte les deux dimensions (performance et surface) pour les deux processeurs considérés, donnez pour chaque configuration de L1 l'efficacité surfacique de chaque processeur.

N.B. : $Efficacitésurfacique = \frac{IPC}{surface(mm2)}$

4. Efficacité énergétique :

D'après [6], les consommations énergétiques du Cortex A7 et du Cortex A15 sont de **0.10 mW/MHz** et **0.20 mW/MHz** respectivement. De plus, en technologie 28 nm, les fréquences maximales du Cortex A7 et du Cortex A15 sont de **1.0 GHz** et **2.5 GHz** respectivement.

Q10 : Quelle puissance en **mW** consomme chaque processeur à la fréquence maximale ?

Q11 : Avec le même protocole que précédemment, et en prenant en compte les deux dimensions (énergie et surface) pour les deux processeurs considérés, donnez pour chaque configuration de L1 l'efficacité énergétique de chaque processeur (à fréquence maximale).

N.B. : $Efficacitéénergétique = \frac{IPC}{consommationenergie(mW)}$

5. Architecture système big.LITTLE :

Q12 : Avec un esprit de concepteur de système, et en se basant sur les résultats de Q10 et Q11, proposez la meilleure configuration du cache L1 du processeur big.LITTLE pour les applications Dijkstra et BlowFish **individuellement.**

6. Facultatif :

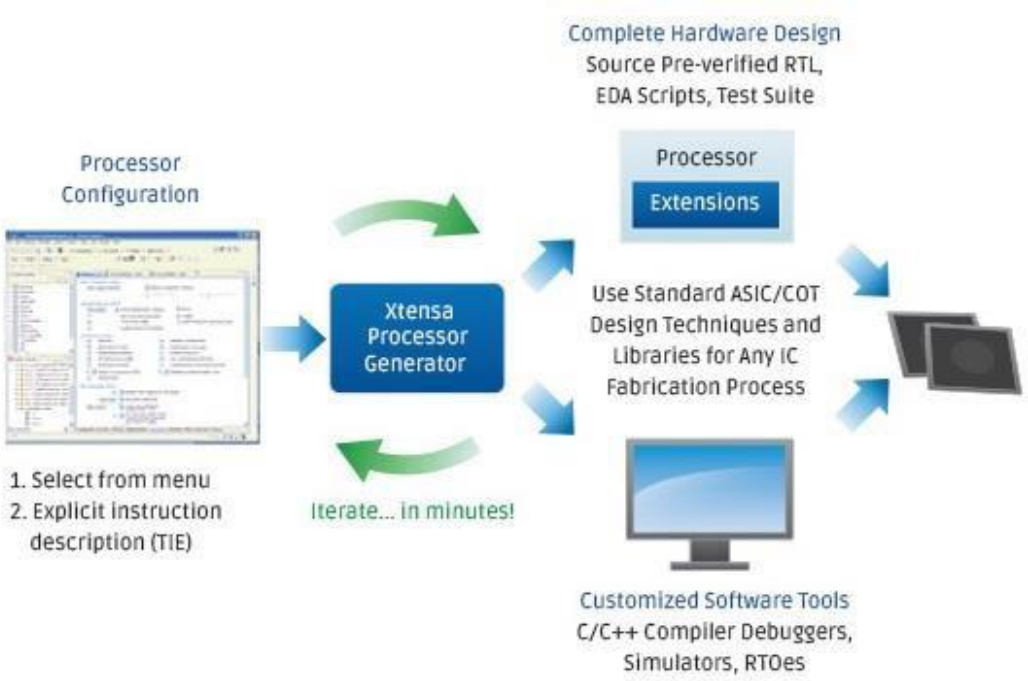
Q13 : Les configurations proposées sont-elles équivalentes ? Proposer éventuellement un compromis et conclure sur les applications étudiées.

Q14 : Proposez une approche pour la spécification d'une architecture avec plusieurs applications dans un domaine spécifique.

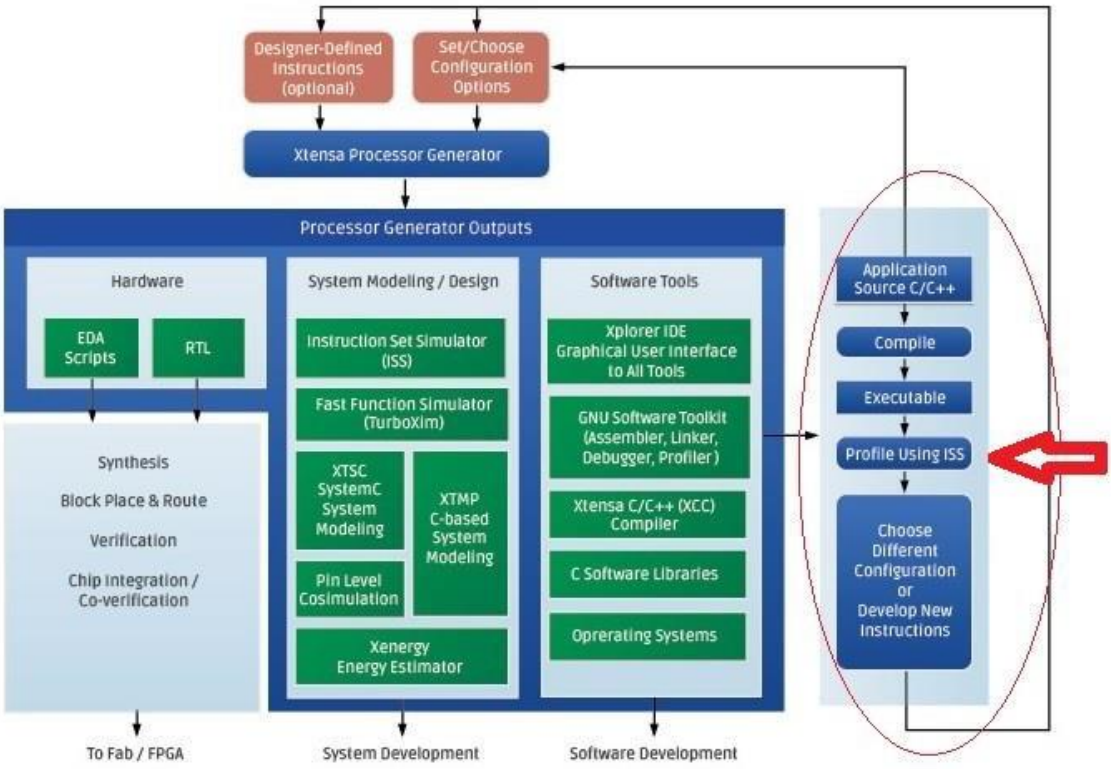
Remettre un rapport de TP par **quadrinôme** en version électronique **au format PDF** (**TP4-nom1-nom2-nom3-nom4.pdf**) incluant l'ensemble des réponses aux questions précédentes pour le **17/03/2025** aux adresses emails hammami@ensta.fr et sasa.radosavljevic@ens-paris-saclay.fr
avec en sujet de message **ECE_4ES01_TA/TP4**.

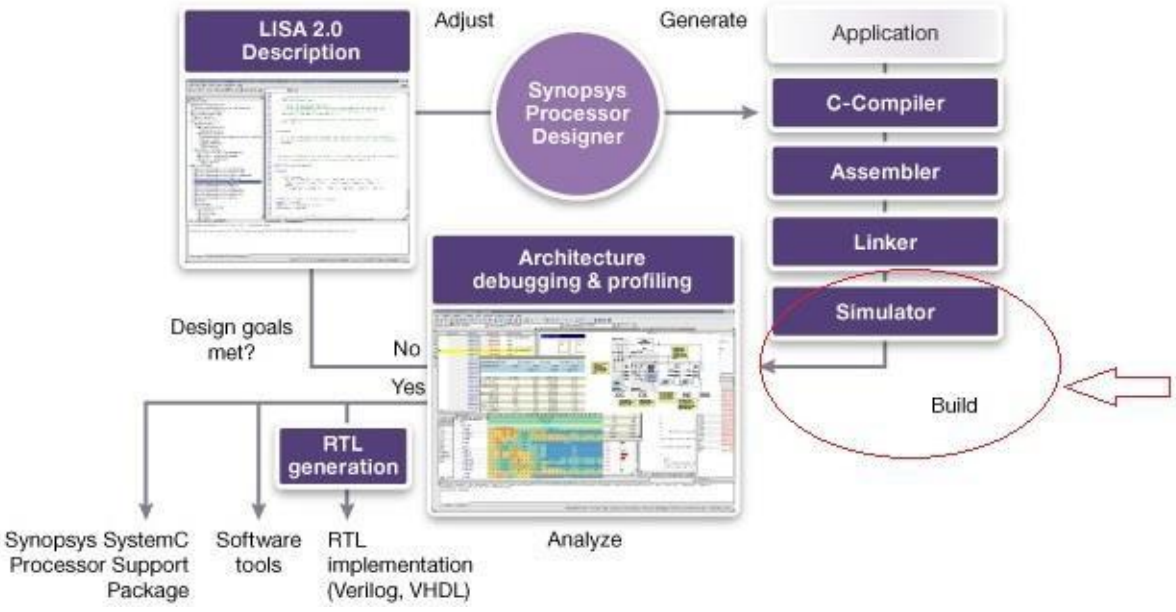
La note du rapport comptera pour 15% de la note globale.

N.B. : *La lisibilité et le format font partie de la note de votre rapport.*



Automated Customization Processor Overview


Designer-Defined Instructions (optional)
Set/Choose Configuration Options
Xtensa Processor Generator
Processor Generator Outputs
Hardware
EDA Scripts
RTL
Synthesis
Block Place & Route
Verification
Chip Integration / Co-verification
System Modeling / Design
Instruction Set Simulator (ISS)
Fast Function Simulator (TurboXim)
XTSC SystemC System Modeling
XTMP C-based System Modeling
Pin Level Cosimulation
Xenergy Energy Estimator
Software Tools
Xplorer IDE Graphical User Interface to All Tools
GNU Software Toolkit (Assembler, Linker, Debugger, Profiler)
Xtensa C/C++ (XCC) Compiler
C Software Libraries
Oprerating Systems
Application Source C/C++
Compile
Executable
Profile Using ISS
Choose Different Configuration or Develop New Instructions
To Fab / FPGA
System Development
Software Development


LISA 2.0 Description
Adjust
Generate
Synopsys Processor Designer
Application
C-Compiler
Assembler
Linker
Simulator
Build
Architecture debugging & profiling
Design goals met?
No
Yes
RTL generation
Analyze
Synopsys SystemC Processor Support Package
Software tools
RTL implementation (Verilog, VHDL)