VISUALISER POUR COMPRENDRE : APPLICATION À LA PROGRAMMATION VECTORIELLE

Thèse présentée comme exigence partielle
du doctorat en informatique cognitive

Par Pierre Marie Ntang

Octobre 2023

REMERCIEMENTS

Pendant mes travaux de thèse, j'ai bénéficié du soutien et de la contribution directe ou indirecte de plusieurs personnes que je remercie et qu'il m'est impossible, compte tenu de l'espace disponible, de nommer individuellement.

Toutefois, je tiens à remercier tout d'abord Daniel Lemire mon directeur de thèse, professeur au Département Science et technologie de l'université TELUQ, sans qui cette thèse n'aurait même pas pu commencer, et encore moins se terminer. Je lui suis reconnaissant pour son soutien multiforme, sa patience et sa capacité à communiquer les bonnes idées et la bonne humeur. Je remercie ensuite Hamadou Saliah-Hassane, professeur au Département Science et technologie de l'université TELUQ pour tous ses conseils et ses marques d'attention. Je remercie mon co-directeur Serge Robert, professeur et directeur de l'Institut des sciences cognitives (ISC) à l'UQAM. Je remercie les membres du jury d'avoir pris le temps d'évaluer mon travail. Je remercie toute l'équipe du Dot-Lab, en particulier les professeurs Wassim Bouachir et Nicolas Bélanger ; mes collègues Blandine, Fatma, Ahmed, Safwen, Hela, Robert et Clément.

Une thèse du fait de ses exigences, induit des sacrifices sur le plan personnel et familial. Je remercie ma famille pour son soutien et sa compréhension. Ainsi, je remercie ma conjointe Louvrance pour sa patience et sa capacité à avoir su suppléer à plein d'égard à mon absence. Je remercie Zhen Aurore, Shen Aubrine, Ivan Avicenne, Iurii Averroes qui m'ont insufflé courage et détermination, et j'espère qu'ils et elles sauront prendre ce travail comme une référence et une exigence pour pousser les frontières de la science plus loin et porter son flambeau plus haut.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

AI	amplification de l'intelligence
DataViz	visualisation de données
IA	intelligence artificielle
InfoViz	visualisation d'information
MIMD	*Multiple Instruction, Multiple Data*
MPMD	*Multiple Program, Multiple Data*
OSF	*Open Science Framework*
OWL	*Web Ontology Language*
PV	*program visualization*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
SIMD	*Single Instruction, Multiple Data*
SPMD	*Single Program, Multiple Data*
SV	visualisation de logiciel
VP	*visual programming*

RÉSUMÉ

Malgré la production de registres vectoriels accompagnés de fonctions adéquates permettant leur utilisation, la programmation vectorielle tarde à s'imposer chez les programmeurs. Pourtant les avantages que ce paradigme de programmation présente sur la programmation scalaire sont indéniables ; on peut citer le gain en performance, le gain économique et même le gain écologique. Le problème semble être la barrière à l'entrée dans ce paradigme. En effet, les programmes vectoriels, même des plus simples et des plus courts, peuvent être rebutants parce que difficiles à comprendre et donc à développer et à maintenir. En ce qui concerne la compréhension de code en général, et l'aide à la programmation parallèle dont la programmation vectorielle est une branche spécifique, une des principales techniques utilisées est la visualisation de logiciel (en abrégé SV pour *Software Visualization* en anglais). Bien que la plupart des études montrent que cette technique est efficace et cognitivement efficiente, elle manque de fondement théorique et de justification scientifique clairs.

Notre thèse se situe à l'intersection de ces deux problèmes. Nous voulons utiliser la visualisation de logiciel (SV) pour abaisser la barrière cognitive à l'entrée de la programmation vectorielle. Nous réalisons cela en facilitant la compréhension des fonctions vectorielles et la compréhension de code écrits à l'aide de ces fonctions, mais dans le cas de la compréhension des fonctions vectorielles, nous capturons auparavant chez l'expert du domaine les connaissances devant nourrir cette compréhension. Tout cela est mis en œuvre dans un prototype, SIMDGiraffe, que nous concevons, affinons et implémentons dans un processus itératif et incrémental. Au cours de ce processus, nous élaborons notamment des modèles de représentation des domaines visualisés, modèles de portée générale. L'étude expérimentale en double aveugle que nous menons permet ainsi non seulement de valider ces modèles, mais contribue à la justification scientifique et au renforcement des fondements théoriques de la SV à travers le risque de falsification qu'elle encourt. Cette étude expérimentale qui montre qu'un novice qui utilise SIMDGiraffe a presque toujours un score en termes de compréhension des fonctions vectorielles supérieure à celui d'un novice qui ne l'utilise pas, soulève des questions quant à l'interaction entre la visualisation et l'explication d'un expert.

Mots clés: visualisation de logiciel, programmation vectorielle, efficience cognitive, modèle.

INTRODUCTION

Des utilisateurs d'applications de jeu aux chercheurs testant les derniers algorithmes d'apprentissage profond, chacun veut toujours une plus grande rapidité dans l'exécution de sa tâche. La recherche de la performance est ainsi une préoccupation permanente pour tous les utilisateurs de logiciels. Il en est de même de l'optimisation de la consommation d'énergie. Compte tenu des taux de pénétration d'internet (66% de la population mondiale utilise internet en 2022) et du téléphone mobile (73% de la population mondiale agée de 10 ans et plus possède un téléphone mobile en 2022) (ITU — International Telecommunication Union, 2023), ces préoccupations concernent presque tout le monde. Cela étant, on peut remarquer que l'exécution d'un programme, quel qu'il soit, se ramène au niveau du processeur à des opérations de base et à des fonctions arithmétiques très simples comme additionner, compter et inverser des bits, etc. Tout gain de performance et d'énergie dans ces opérations est donc un gain pour l'ensemble des programmes. Pour gagner en performance, on peut traiter la même quantité de données, mais plus vite, ou alors on peut traiter de plus grandes quantités de données à la même vitesse, ou enfin on peut idéalement traiter plus vite de plus grandes quantités de données. Les architectures parallèles peuvent permettre d'exploiter ces possibilités de plusieurs façons, avec en prime un gain énergétique. Ainsi, les architectures *Multiple Instruction, Multiple Data* (MIMD) exploitent ces possibilités, en offrant plusieurs unités de traitement qui exécutent différentes instructions sur différents flux de données en même temps. Les architectures *Single Instruction, Multiple Data* (SIMD) permettent quant à elles d'exécuter la même instruction sur une plus grande quantité de données. Le parallélisme SIMD, bien qu'étant la

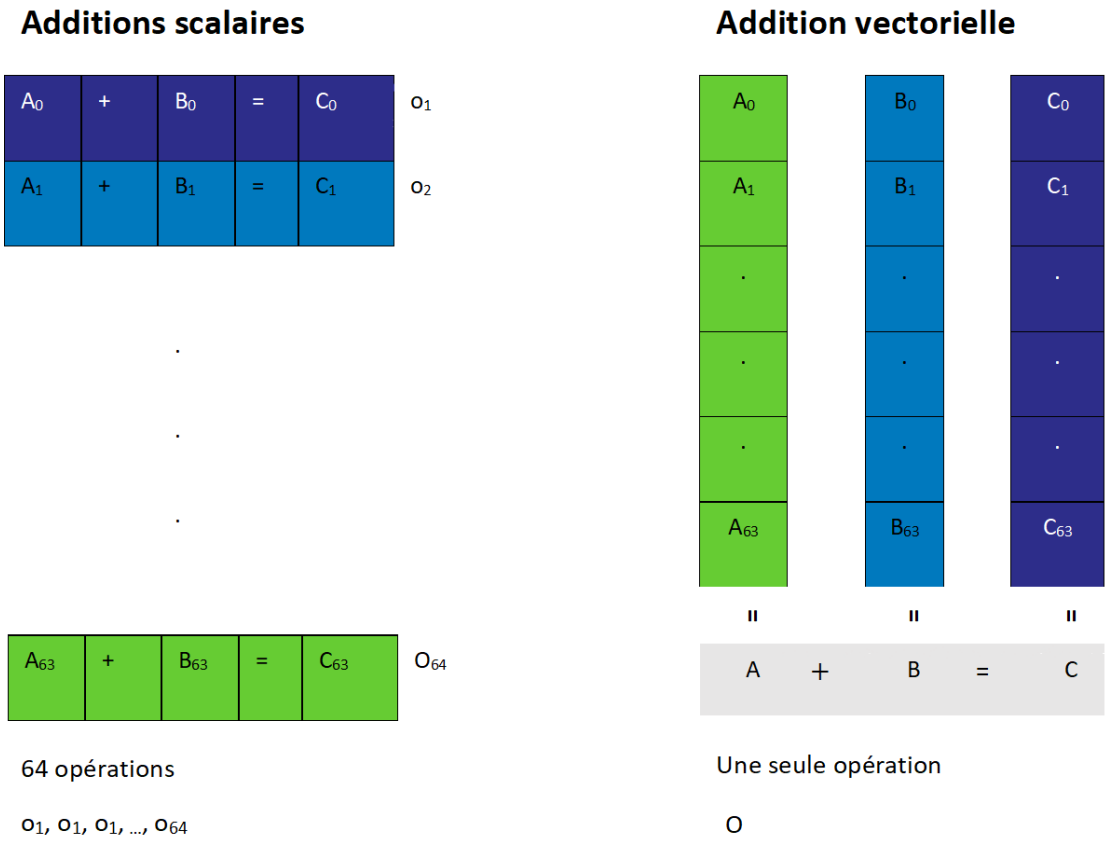


FIGURE 0.1 – Addition deux à deux de 128 mots-entiers à l'aide de 64 opérations scalaires à gauche contre une opération vectorielle à droite.

plus ancienne forme d'architecture parallèle avec ILLIAC IV (Barnes *et al.*, 1968) a retrouvé un regain d'intérêt récemment. Ainsi, en 2015 le fabricant Intel® a mis sur le marché son architecture AVX-512 (Schmidt *et al.*, 2018) qui succède ainsi, pour ce qui est des architectures vectorielles d'Intel®, au MMX 64 bits. Dans les architectures SIMD, le parallélisme est mis en œuvre par le recours à la vectorisation des données ; c'est pourquoi ce type d'architecture est dite vectorielle et le paradigme de programmation qu'elle induit est appelé programmation vectorielle par opposition aux architectures et à la programmation dites scalaires. À titre d'illustration, supposons qu'on veuille additionner deux à deux 128 nombres

entiers $A_0, \ldots, A_{63}, B_0, \ldots, B_{63}$ avec pour résultats $C_0, \ldots, C_{63}$ comme indiqué sur la Figure 0.1. On a donc $A_i + B_i = C_i$. Sur une architecture scalaire, on procède par 64 opérations d'additions scalaires comme indiqué sur la partie gauche de la Figure 0.1. Sur une architecture vectorielle, en une seule opération vectorielle, on obtient le même résultat comme indiqué sur la partie droite de cette même Figure 0.1. Si on suppose que pour $i \in \{0, \ldots, 63\}$ o_i dure t_i et consomme une énergie e_i ; que O dure T et consomme une énergie E, on a $T \ll \sum_{i=0}^{63} t_i$ et $E \ll \sum_{i=0}^{63} e_i$ (Cebrian *et al.*, 2020). Ainsi, l'architecture SIMD permet de consommer beaucoup moins d'énergie et est donc plus économique et écologique, en plus du gain de performance qu'elle apporte.

Cependant, la programmation vectorielle n'est pas aisée, c'est une tâche rebutante. En fait, pour tirer pleinement parti de ces architectures, il y a essentiellement deux possibilités. La première possibilité consiste à maîtriser et utiliser le jeu d'instruction fourni par le fabricant ; ce qui revient à programmer en assembleur. Cela n'est pas du tout évident, car alors, en dehors des instructions vectorielles proprement dites, il faut gérer soit même les registres, les positionnements de drapeaux (flags en anglais), etc. La seconde possibilité consiste à maîtriser et utiliser les fonctions vectorielles ou *intrinsics*. En effet, conscients à la fois du potentiel de l'architecture SIMD et du caractère rebutant de la programmation vectorielle, les fabricants de matériels accompagnent le jeu d'instructions d'un ensemble de fonctions vectorielles dont l'expression syntaxique est plus proche de celle des fonctions d'un langage de plus haut niveau comme le C/C++ : ce sont les *intrinsics*. Toutefois, cette proximité avec le langage C/C++ ne devient évidente que lorsqu'on a déjà une certaine expertise dans la connaissance et la manipulation des *intrinsics*. De plus, même pour un expert de la programmation SIMD, il n'est pas toujours évident de comprendre le comportement d'un code produit suivant le paradigme de programmation vectorielle. Il importe donc, si l'on veut que le

potentiel de cette architecture soit actualisé par la communauté des développeurs, de trouver un moyen d'abaisser le niveau de difficulté à l'entrée de ce paradigme de programmation, mais aussi de faciliter la compréhension du comportement d'un code produit suivant ce même paradigme.

Or, en observant aussi bien les fabricants de matériel de l'architecture SIMD que les experts qui écrivent des programmes vectoriels, on remarque que ces acteurs ont souvent recours à des graphiques et à des couleurs dans leurs explications du fonctionnement des *intrinsics* et du fonctionnement de codes écrits avec ces *intrinsics* ou même écrits avec les instructions vectorielles. L'utilisation de ces techniques de visualisation—car c'est bien de ça dont il s'agit—, quoique très utile présente beaucoup de limites. D'abord l'efficacité des effets visuels obtenus sont entièrement tributaires de l'intuition de leurs auteurs ; de plus ces effets visuels sont susceptibles d'être biaisés en raison de l'effet *expert blind spot*. L'effet *expert blind spot* consiste pour un expert à donner une explication qui est inaccessible à un non-expert parce que l'explication en question se préoccupe plus de l'expertise expliquée que de la compréhension de cette explication : c'est une explication biaisée en faveur d'un expert (Nathan *et al.*, 2001). Enfin les effets visuels en question et les artifices mobilisés pour les produire ne visent que les cas particuliers concernés. Malgré les limites mentionnées, le recours à ces effets visuels peut être considéré, quoiqu'à un niveau rudimentaire, comme de la SV. En tout cas, ce recours exprime un besoin et désigne la SV comme un moyen d'y pourvoir.

La visualisation de logiciel (SV) est l'utilisation de signes iconiques ou de tout autre procédé métaphorique pour évoquer chez le destinataire des images mentales susceptibles de l'aider à comprendre tout artefact lié à un programme informatique ou au processus de développement de ce programme (Diehl, 2007). La SV fait partie d'un domaine scientifique plus large qui est la visualisation d'information (en abrégé InfoViz pour *Information Visualization en anglais*). Ce

domaine scientifique est très actif, que ce soit dans la recherche de solutions à des problèmes concrets (*Design Study*), la conception de nouveaux algorithmes de visualisation ou même le questionnement sur ses propres fondements en tant que science (Correll, 2022; Purchase *et al.*, 2008). Parmi les aspects d'un programme auxquels s'intéresse la SV, on peut distinguer sa structure, son comportement et enfin son évolution. La SV est déjà utilisée pour comprendre le comportement (ou le fonctionnement) de codes parallèles (Papenhausen *et al.*, 2016; Wilhelm *et al.*, 2016; Isaacs *et al.*, 2014; Li *et al.*, 2017). Cependant, à notre connaissance, les outils de la SV n'ont pas encore été mobilisés pour essayer de résorber les difficultés liées à la programmation vectorielle et c'est là l'originalité de deux de nos importantes contributions : utiliser la SV pour faciliter la compréhension du fonctionnement des fonctions vectorielles (ou *intrinsics*), mais aussi la compréhension du comportement d'un code produit avec ces fonctions lors de son exécution sur une architecture cible donnée.

Il faut souligner que faisant partie du champ plus large de la visualisation d'information (InfoViz), la SV peut bénéficier des acquis des domaines de ce champ parent ; encore faudrait-il pouvoir transposer ces acquis de l'InfoViz à la SV. Si cela peut se faire aisément en ce qui concerne la visualisation de la structure du code et la visualisation de l'évolution du code, il en va autrement pour ce qui a trait à la visualisation du comportement de code. En effet, la transposition des outils et des méthodes de l'InfoViz à la SV pour ce qui a trait à la visualisation du comportement de code n'est pas du tout triviale. Une des raisons à cette non-trivialité est que les données—qu'il faut produire—font partie intégrante du problème de la SV en ce qui concerne le comportement de code (Ntang et Lemire, 2021). Même si la mobilisation de ces outils pour résoudre des problèmes concrets est la principale préoccupation de l'InfoViz et par conséquent de la SV, cela n'occulte pas les problèmes sur ses fondements qui continuent de se poser

pour ce domaine de pratique en tant que discipline scientifique ou programme de recherche au sens de Lakatos (1980). Ainsi l'InfoViz (et par ricochet la SV) peut paraître par certains aspects comme une discipline scientifique en crise. Certains se posent même la question sur sa nécessité ou sa pérennité (Lorensen, 2004; Sedig et Parsons, 2016; van Wijk, 2006). Le fait est que, focalisés la plupart du temps sur la valeur d'usage de leur apport, les chercheurs en InfoViz se préoccupent en général moins de la validité de leur pratique scientifique, c'est-à-dire de leur science. Or aucun programme de recherche scientifique ne peut faire l'économie de cette préoccupation sur sa validité, au risque d'être condamné à la dégénérescence. Un des aspects de ce biais en faveur de préoccupations utilitaristes est le faible pourcentage d'études d'évaluation ou de validation rigoureuse des outils développés en SV. Ainsi, jusqu'à 62% de ces outils ne font l'objet d'aucune évaluation ou validation (Merino *et al.*, 2018). Mais même lorsqu'il y a une étude d'évaluation ou de validation, il s'agit d'ordinaire de validation anecdotique, de scénario d'utilisation, etc. (Chotisarn *et al.*, 2020) ; toute chose qui est loin de constituer une preuve de corroboration entre le discours théorique de la SV et ses observables. Dans ce contexte, la question de savoir si la SV aide réellement à comprendre la structure, l'évolution et le comportement d'un programme ou de ses artefacts reste donc légitime et actuelle. Une réponse directe à cette question est impossible en raison de son caractère général et universel (Popper, 2005; Lakatos, 1976). Mais on peut faire progresser la discipline scientifique en identifiant et en éliminant de potentiels falsificateurs par la corroboration. Donc, quand bien même cette question aurait été tranchée dans d'autres domaines des observables de la SV, la poser dans le domaine de la programmation vectorielle reste pertinente et d'actualité. D'où l'originalité de l'une de nos contributions : Est-ce que la SV permet réellement d'aider à comprendre le comportement d'une fonction vectorielle ? Plus précisément, est-ce qu'il y a une différence significative dans la compréhension du fonctionnement d'un *intrinsic* entre ceux qui utilisent un outil reposant sur les

prescriptions de la SV (SIMDGiraffe) et ceux qui n'utilisent pas cet outil ? Suivre les prescriptions de la SV dans le développement de SIMDGiraffe revient en effet à rendre possible l'application des hypothèses aux conséquences prédictives de cette science au domaine de la programmation vectorielle ; de la compréhension des fonctions vectorielles plus exactement. Et d'après ce qui découle de ces hypothèses aux conséquences prédictives de la SV, la différence observée devrait se traduire par une plus grande compréhension des fonctions vectorielles par ceux qui utilisent SIMDGiraffe.

Pour voir ce qu'il en est réellement de ces hypothèses de la SV et des prédictions qui en découlent par voie de conséquence, mais aussi pour parcourir l'ensemble des points évoqués, nous structurons notre démarche ainsi : au chapitre 1, nous présentons notre problématique et articulons nos objectifs et notre question générale de recherche. Au chapitre 2 à travers la revue de littérature, nous fixons le cadre théorique de notre travail. Nous sortons de cette revue de littérature avec nos questions de recherche et la clarification des hypothèses qui fondent et structurent notre démarche des points de vue respectifs de la modélisation de comportement de code et de la SV. Au chapitre 3, en mobilisant les outils de la SV, et après avoir développé les modèles adéquats des domaines respectifs nous implémentons les deux modules de SIMDGiraffe. Ces modèles sont d'importantes contributions ne serait-ce que du point de vue de l'InfoViz (Munzner, 2008). Le chapitre 4 commence par l'élaboration de la méthodologie de l'expérimentation pour réfuter le risque de falsification de la SV en ce qui concerne la compréhension du fonctionnement d'un *intrinsic*. Nous présentons ensuite les données collectées lors de cette expérimentation, les analyses et les principaux résultats qui en découlent. Nous terminons enfin notre démarche par une conclusion et des perspectives. En annexe, nous avons des documents comme le certificat d'éthique, les questionnaires utilisés et les lettres de sollicitation envoyées.

CHAPITRE I

PROBLÉMATIQUE ET OBJECTIFS

Dans ce chapitre, nous allons d'abord présenter les problèmes qui sont à l'origine de notre démarche et qui touchent donc nos intérêts de recherche, c'est-à-dire les motivations de ces travaux de recherche. Nous structurons ces motivations en deux parties : la partie 1.1.1, motivation scientifique qui repose sur des considérations théoriques, et la partie 1.1.2, motivation pratique qui repose sur des considérations plus concrètes, en termes d'utilité pratique. Nous clarifions ensuite ces intérêts de recherche en articulant l'objectif général ou global de notre démarche que nous développons en différents objectifs secondaires et spécifiques qui le recouvrent. Nous terminons enfin ce chapitre par la section 1.3, Question générale de recherche, où nous formulons la question générale de recherche que nous déclinons sous ses différents aspects.

1.1 Motivations

Compte tenu des limites de la vectorisation implicite (Amiri et Shahbahrami, 2020), et de ce que de toute façon ceux qui développent les librairies et les API qui sont des préalables à la vectorisation implicite doivent faire de la programmation vectorielle explicite ; compte tenu du fait que la programmation explicite à l'aide des instructions vectorielles revient à programmer en assembleur, ce qui

est extrêmement difficile, lourd et exigeant (Stojanov *et al.*, 2018), la programmation vectorielle explicite avec l'utilisation des fonctions vectorielles (*intrinsics*) apparaît comme le meilleur choix pour tirer profit des architectures SIMD. Mais puisque comprendre les fonctions vectorielles ou du code produit avec ces fonctions reste quand même très difficile par rapport à la programmation ordinaire, c'est-à-dire la programmation scalaire, il est important d'aider à aplanir ces difficultés au regard des bénéfices (Barredo *et al.*, 2020; Hennessy et Patterson, 2011) qu'on attend du code vectoriel. Les outils de la SV s'avèrent tout indiqués pour aider à aplanir ces difficultés. Mais nous savons que ces outils font encore l'objet des travaux et de discussion (Purchase *et al.*, 2008; Sedig et Parsons, 2016; Chen *et al.*, 2016; Bedu *et al.*, 2019; Correll, 2022) en ce qui concerne leur justification au sens épistémologique.

1.1.1 motivation scientifique

Un des problèmes de ces travaux et de ces discussions autour de la justification épistémologique des outils de la SV est qu'ils se font en dehors de tout cadre expérimental : ils manquent ainsi d'ancrage dans la réalité, ce qui est susceptible d'ébranler la validité de la SV en tant que science. Or la justification épistémologique porte précisément sur cette validité, et cela en essayant de répondre à la question de la correspondance de l'objet théorique de la pratique scientifique de la SV avec son objet réel. On sait que, parce que la vérité est inaccessible (Popper, 2005; Lakatos, 1976), une telle réponse ne peut se faire que par la falsification ; c'est-à-dire en essayant de falsifier ou de corroborer cet objet théorique par des cas concrets. Ainsi la vitalité d'une pratique scientifique c'est-à-dire la non-dégénérescence du programme scientifique (Lakatos, 1980) auquel correspond cette pratique se mesure donc concrètement par le nombre de falsificateurs potentiels non falsifiables *a posteriori*, parce qu'ayant corroboré le programme en

question. De ce point de vue, il y a du travail à faire dans la pratique scientifique en ce qui concerne la SV. De fait, ceux qui conçoivent et réalisent les systèmes de visualisation le font en général pour répondre à un besoin pratique. Une grande majorité de ces systèmes (jusqu'à 62%) ne font même pas l'objet d'une évaluation rigoureuse (Merino *et al.*, 2018). Mais même quand il y a une évaluation considérée comme rigoureuse, elle porte sur la satisfaction des bénéficiaires et/ou des utilisateurs du système (Chotisarn *et al.*, 2020) ; or cette satisfaction est une mesure nécessairement subjective, et donc sujette à des biais. En effet étant donné un outil de visualisation dont l'objectif est d'aider l'utilisateur à effectuer une tâche quelconque, la quasi-totalité des outils à notre connaissance ont pour méthode d'évaluation rigoureuse des études de cas ou des scénarios d'utilisations pouvant inclure ou non des utilisateurs autres que les développeurs. Toutes ces méthodes d'évaluations reviennent soit à mesurer la satisfaction affirmée ou constatée de l'utilisateur/bénéficiaire, soit la satisfaction du chercheur en SV à l'origine du système évalué après une analyse plus ou moins rigoureuse de ce système (Merino *et al.*, 2018). Il nous semble évident qu'aucune de ces satisfactions n'est la preuve que le système de visualisation développé en mobilisant les outils de la SV améliore la performance de l'utilisateur dans l'exécution de sa tâche, peu importe comment cette performance est calculée. Notons tout de même que certains outils se détachent en termes de rigueur dans l'évaluation que leurs auteurs font du système développé ; par exemple Farghally *et al.* (2017) utilisent un groupe de contrôle et un groupe expérimental pour mener l'évaluation de l'apport de leur système de visualisation (*AAVs pour Algorithm Analysis Visualizations*) dans la performance des étudiants par rapport à la compréhension d'algorithmes. Néanmoins, un des reproches qu'on peut faire à la démarche des auteurs est le décalage dans le temps entre le groupe de contrôle et le groupe expérimental, l'un des groupes étant évalué en automne, et l'autre au printemps. Ce décalage, même s'il n'enlève rien à la validité des conclusions de l'étude, augmente les risques de biais inconnus

corrélés à la saison par exemple. Au demeurant, en remarquant que le domaine de la programmation SIMD n'a pas encore à notre connaissance fait l'objet d'études visant l'utilisation des outils de la SV pour en faciliter la compréhension, même si la rigueur des différentes études était absolument parfaite dans d'autres domaines, la question de la corroboration de la pratique scientifique de la SV dans le domaine de la programmation SIMD reste encore importante et intéressante.

1.1.2 motivation pratique

Au-delà de la démarche visant la corroboration des outils de la SV dans le domaine de la programmation SIMD, la mobilisation de ces outils pour faciliter la compréhension des fonctions vectorielles (*intrinsics*) et de codes produits à l'aide de ces fonctions est en soi importante, et le caractère pionnier de cette mobilisation pour ce domaine renforce cette importance et constitue une des originalités de notre démarche et de notre contribution à la fois à la SV et à la programmation SIMD. Au bout de cette démarche, la production d'un prototype fonctionnel nommé SIMDGiraffe est une motivation supplémentaire. Au regard de ces différentes motivations, nous pouvons maintenant formuler notre objectif de recherche.

1.2 Objectif

L'objectif principal et général de cette thèse est d'abaisser la barrière (cognitive) à l'entrée de la programmation vectorielle, et subséquemment de contribuer à la justification scientifique et au renforcement des fondements théoriques de la SV. Cet objectif général se décline en trois objectifs spécifiques qui sont de développer des modèles de domaines comme support à la présentation visuelle, mais des modèles de portées tout à fait générales et allant donc au-delà de la SV ; et enfin de valider l'ensemble de la démarche par une étude expérimentale. Plus précisément

ces trois objectifs spécifiques peuvent s'énoncer ainsi :

1.2.1 Premier objectif spécifique

Il est question dans le cadre de cet objectif de développer un modèle du domaine des fonctions vectorielles. Ce modèle sert de base de la représentation des données dans le module 2 du prototype SIMDGiraffe que nous concevons et implémentons. La possibilité de cette représentation basée sur ce modèle, et avec les données concrètes du domaine des fonctions vectorielles passe par une vérification du modèle développé en testant sa cohérence ; cela constitue aussi une (première) validation du modèle, car cette possibilité prouve sa consistance par rapport au domaine en question (Bjøner, 2006). Du point de vue de la méthodologie suivie (Munzner, 2009; Diehl, 2007), rappelons que la possibilité de construire une telle présentation basée sur le modèle est une seconde validation du modèle en question. Bien sûr, la validité du modèle est éventuellement renforcée plus tard si l'outil développé permet de réaliser l'objectif pour lequel cet outil est conçu et implémenté. La présentation visuelle construite à partir de cette représentation vise à aider le programmeur à comprendre le comportement des fonctions vectorielles, en particulier les fonctions vectorielles correspondant au jeu d'instructions de l'AVX-512 du fabricant Intel®. La portée du modèle développé va toutefois au-delà de la visualisation et nous donnons quelques indications sur comment ce modèle peut être exploité dans d'autres contextes. Pour cet objectif spécifique, nous nous inspirons de la possibilité contenue dans l'hypothèse de Dasgupta *et al.* (2019) de pouvoir décomposer un code en des actions simples. Mais, nous appliquons cette possibilité non plus à une architecture physique cible donnée, mais à un niveau d'abstraction un peu plus élevé ; c'est-à-dire plus proche de l'algorithmique. La tâche correspondante se situe entre l'étape 1 et l'étape 2 de la méthodologie de Munzner (2009).

1.2.2 Second objectif spécifique

Pour cet objectif, il est question de développer un modèle décrivant l'exécution d'un code sur une architecture donnée. Ce qui est dit pour le premier objectif reste valable pour le second objectif *mutatis mutandis*. Appliqué aux codes produits à l'aide de fonctions vectorielles ou *intrinsics*, ce modèle sert de base à la représentation de données dans le module 1 du prototype SIMDGiraffe que nous concevons et implémentons. La présentation visuelle construite à partir de cette représentation vise à aider le programmeur à comprendre le comportement de codes, plus particulièrement le comportement de fonctions développées à l'aide de fonctions (*intrinsics*) ou d'instructions vectorielles, notamment les fonctions vectorielles correspondant au jeu d'instructions du fabricant Intel® ; il s'agit des jeux d'instructions MMX, SVML, etc., et de ceux des familles SSE, AVX et AVX-512. Cet objectif repose en partie sur l'hypothèse de Dasgupta *et al.* (2019) à savoir la possibilité de décrire un code s'exécutant sur une architecture cible donnée par des actions simples comme lire, effectuer des opérations élémentaires, écrire. La tâche correspondante se situe entre l'étape 1 et l'étape 2 de la méthodologie de Munzner (2009).

1.2.3 Troisième objectif spécifique

Concevoir et exécuter une étude expérimentale pour vérifier la réalisation de la première partie de l'objectif général, mais aussi pour dégager des éléments permettant de répondre à la seconde partie de cet objectif principal. La méthodologie suivie lors de cette étude expérimentale et le retour d'expérience y afférent permet d'enrichir la pratique épistémologique dans le domaine de la SV en termes de validation des outils de visualisation comme le laissent suggérer Borgo *et al.* (2018). Afin que cette étude expérimentale soit exempte de tout biais subjectif comme

peuvent l'être les études de cas, les scénarios de cas d'utilisation, etc., couramment utilisés en visualisation de logiciel, nous la menons en double aveugle et avec des participants aléatoirement distribués dans les différents groupes à l'étude : c'est donc une étude de validation randomisée en double aveugle. Dans la réalisation de cet objectif, nous pouvons nous appuyer sur l'aide-mémoire et la liste des bonnes pratiques de Borgo *et al.* (2018) qui présente pourquoi et comment on doit tirer partie de la possibilité de recruter des participants en ligne pour évaluer et valider les projets d'InfoViz.

1.3 Question générale de recherche

Nous pouvons cristalliser tous ces objectifs et ces motivations en la question générale de recherche suivante que nous élucidons progressivement ensuite : la visualisation peut abaisser la barrière cognitive à l'entrée de la programmation vectorielle (ou programmation SIMD). Autrement dit, on peut utiliser le langage visuel pour aider [les programmeurs] à embrasser le paradigme de programmation vectorielle en diminuant le supplément de difficulté qu'il représente par rapport au paradigme de programmation scalaire. Plus exactement, nous pouvons nous servir du langage iconique pour aider à la compréhension du comportement des fonctions vectorielles (intrinsics) aussi bien que des codes écrits en utilisant ces fonctions. Dans la suite de cette section, nous utilisons le terme code pour désigner indistinctement un *intrinsic* ou un code source produit en utilisant des *intrinsics*. Pour nous servir du langage iconique, nous pouvons nous appuyer sur les acquis des techniques et méthodes de visualisation de la SV, et plus généralement de l'InfoViz dont la SV est une branche. Mais comme le remarquent Sedlmair *et al.* (2012), l'utilisation de la visualisation ne devient une démarche scientifique qu'à partir du moment où cette démarche est consubstantielle à une réflexion introspective sur elle-même. Cette réflexion peut être l'opportunité de se pencher sur la pratique

épistémologique de la SV. Il convient aussi de noter que la visualisation du comportement d'un code ne peut pas être obtenue en traitant au sens de l'InfoViz (et par ricochet de la SV) ce code ; d'où se pose la question du passage du code aux données devant décrire son comportement. Il y a donc la nécessité de trouver un modèle permettant d'abstraire à partir du code son comportement en prenant en compte d'autres paramètres tels que l'environnement d'exécution qui peut être réel comme lors d'une exécution sur une machine physique, ou plus abstrait, dans la pensée d'un sujet humain, comme lors d'une exécution abstraite. Bien sûr, le résultat de ces modèles doit être des représentations, bref des données décrivant le comportement du code en question. Pour affiner et opérationnaliser cette question générale de recherche, et ce dans l'optique d'y apporter les réponses appropriées dans le cadre de la réalisation de nos objectifs de recherche, il convient d'explorer la littérature adéquate.

CHAPITRE II

REVUE DE LA LITTÉRATURE, CADRE THÉORIQUE ET HYPOTHÈSES

Dans ce chapitre, après avoir montré pourquoi le paradigme de programmation qui découle des architectures parallèles en général et des architectures SIMD en particulier mérite d'être compris et adopté, nous présentons les difficultés qui entravent cette compréhension et cette adoption par les programmeurs. Ces difficultés sont pour l'essentiel considérées relativement au paradigme de programmation scalaire. Nous évoquons ensuite quelques approches permettant d'aider à surmonter ou à contourner ces difficultés, et parmi ces approches, nous expliquons pourquoi celles reposant sur la visualisation de logiciel sont indiquées, et donc pourquoi nous les retenons pour aider à surmonter ces difficultés. Nous passons en revue les possibilités offertes par ces approches reposant sur la visualisation de logiciel et les problèmes qu'elles posent dans la pratique, mais aussi en tant que domaine scientifique. En remarquant que toutes ces questions informatiques abordées ont des aspects cognitifs, nous structurons ce chapitre en cinq sections : la section 2.1 porte sur le Cadre informatique et la section 2.2 porte sur le Cadre cognitif. La section 2.3 porte sur La visualisation de logiciel : méthodes, moyens et problèmes. Dans cette dernière section, nous présentons d'abord les méthodes et moyens mis en œuvre en visualisation de logiciel en particulier, mais aussi pour certains, en visualisation de l'information de façon plus générale ; nous présentons ensuite les problèmes que peut poser le recours à la visualisation de logiciel. Dans la sec-

tion 2.4, nous examinons les différents Modèles de comportement de code et pourquoi il est nécessaire d'en développer de nouveaux dans le cadre de nos travaux. Enfin, nous terminons ce chapitre par la section 2.5 qui porte sur les Hypothèses et questions de recherches. Dans cette section nous soulignons les hypothèses qui rendent possible notre démarche. Nous soulignons également l'ensemble des questions opérationnelles reposant sur ces hypothèses et auxquelles notre démarche vise à apporter des réponses.

2.1 Cadre informatique

La composante matérielle est l'un des paramètres dont dépend la performance d'un système de traitement de l'information, lequel système nous assimilons à un ordinateur. L'élément essentiel du matériel dont dépend la performance d'un ordinateur est le processeur. L'autre paramètre dont dépend la performance d'un ordinateur est naturellement le logiciel. Nous utilisons le terme logiciel pour désigner tout code informatique pouvant s'exécuter. Pour un logiciel s'exécutant sur un ordinateur donné, la performance se mesure en temps d'exécution et en espace mémoire utilisé. Pendant longtemps, l'amélioration de la performance a reposé presque uniquement sur l'amélioration de la fréquence d'exécution qui dépend du cycle d'horloge. Mais aujourd'hui, les possibilités d'amélioration de ces caractéristiques physiques sont peut-être épuisées et l'exigence d'ordinateurs plus écologiques est plus forte. La réalisation de ce double défi performance et écologie repose actuellement sur l'organisation des éléments matériels plutôt que sur l'amélioration des caractéristiques physiques de ces éléments (Schmidt *et al.*, 2018). C'est pourquoi les architectures parallèles et les paradigmes de programmation qu'elles sous-tendent sont devenus indispensables.

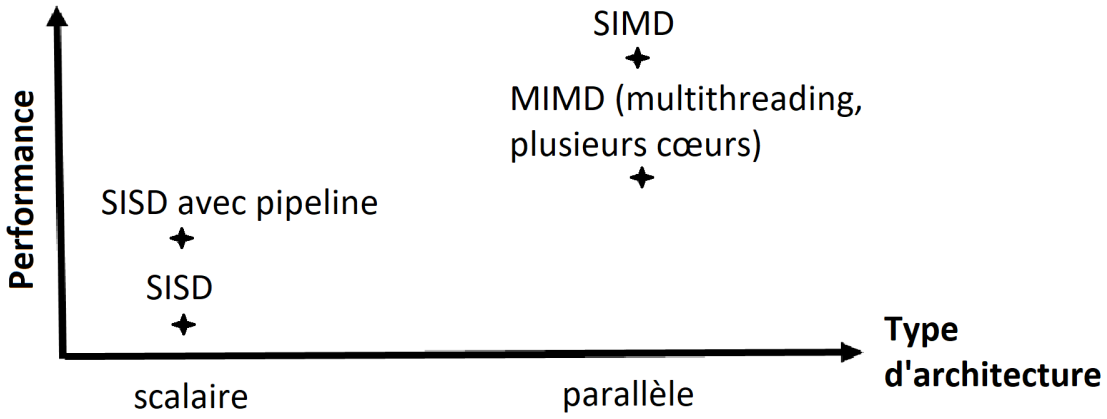
2.1.1 Les architctures parallèles

Pour avoir un aperçu de ces paradigmes et de leur complexité relative, on peut les décrire en partant de la taxonomie de Flynn (Figure 2.1). Cette taxonomie comprend quatre types d'architectures :

- SISD (*Single Instruction, Single Data*) fait référence à l'architecture traditionnelle de von Neumann où un seul traitement séquentiel par une seule unité de traitement fonctionne sur un seul flux de données (peut être amélioré avec une stratégie de pipeline).
- MIMD (*Multiple Instruction, Multiple Data*) fait référence à une architecture où plusieurs unités de traitement exécutent différentes instructions sur différents flux de données.
- SIMD (*Single Instruction, Multiple Data*) fait référence à une architecture où une même unité de traitement effectue la même opération sur plusieurs éléments de données simultanément.
- MISD (*Multiple Instruction, Single Data*) fait référence à une architecture où plusieurs unités de traitement exécutent différentes instructions sur un seul flux de données. Ce type de parallélisme n'est pas si courant, mais peut être trouvé dans les architectures pipelinées.

2.1.1.1 Les architectures MIMD

Les architectures MIMD correspondent à la forme de parallélisme la plus répandue, même si elle n'est pas la plus ancienne (Quinn, 2003). On peut diviser les architectures MIMD en deux branches : les architectures *Single Program, Multiple Data* (SPMD) et les architectures *Multiple Program, Multiple Data* (MPMD). Bien que les termes *multithreading* et *multiprocessing* propres à l'architecture MIMD soient souvent confondus et assimilés au terme même de programmation parallèle,



Les types, sur l'axe horizontal, suivent une complexité architecturale matérielle croissante.

FIGURE 2.1 – Performance selon les différents types d'architecture, inspiré de Schmidt *et al.* (2018).

il convient de noter que le *multithreading* correspond davantage à un partage de ressource de manière à optimiser les performances dans les programmes exécutés. Ce partage peut se faire sur une même unité de traitement (un cœur) d'un processeur. Le *multiprocessing* correspond quant à lui à la parallélisation de l'exécution d'un programme sur plusieurs cœurs d'un ou plusieurs processeurs afin d'accroître les performances d'exécution. La programmation parallèle est assez difficile, en tout cas plus exigeante que la programmation scalaire ou séquentielle (Quinn, 2003). Plusieurs approches ont été ainsi explorées pour faciliter l'adoption du paradigme parallèle (MIMD) par les programmeurs. On peut mentionner les possibilités d'étendre les compilateurs existant pour traduire des programmes séquentiels en programmes parallèles ; d'étendre les langages existant avec de nouvelles opérations qui permettent aux programmeurs d'exprimer le parallélisme ; d'ajouter une nouvelle couche de langage parallèle au-dessus de langages séquentiels existants ; et enfin de définir un tout nouveau langage parallèle et un nouveau système de compilation. Actuellement, le choix va plutôt vers plusieurs standards dont les

plus adoptés sont MPI et OpenMP (Quinn, 2003). Ces standards correspondent à l'extension de langages séquentiels. MPI est une bibliothèque de fonctions qui permet aux processeurs qui n'ont pas de mémoire partagée de communiquer par message. OpenMP est un ensemble de directives de compilation regroupées dans une librairie et qui permettent au compilateur de générer du code *multithread* pouvant tirer parti de plusieurs processeurs disponibles dans un multiprocesseur à mémoire partagée. Les deux standards sont donc complémentaires. Il est intéressant de noter que le standard OpenMP et d'autres comme CUDA, OpenCL, OpenACC sont repris dans le cadre des architectures SIMD.

2.1.1.2 Les architectures SIMD

Bien qu'historiquement la première forme de parallélisme (Quinn, 2003), c'est tout récemment que les architectures SIMD ont connu un regain d'intérêt. L'AVX-512 d'Intel® a été ainsi mis sur le marché seulement en 2015 (Schmidt *et al.*, 2018). Les architectures SIMD permettent un parallélisme de données, c'est-à-dire qu'elles permettent d'exécuter la même instruction sur plusieurs flux de données. Un des avantages que ces architectures présentent par rapport aux architectures MIMD est la présence d'un seul bloc de contrôle (Schmidt *et al.*, 2018). Du reste, les architectures SIMD sont très efficaces en termes de performance et très efficientes en termes de consommation d'énergie et sont aujourd'hui disponibles jusque sur les terminaux mobiles. Cette efficience et cette efficacité sont proportionnelles à un facteur qui dépend de la taille des registres (Cebrian *et al.*, 2020). L'efficience énergétique, au-delà de la satisfaction économique, répond à une exigence écologique. Nous réservons dans cet exposé le terme vectorisation pour le parallélisme lié aux architectures SIMD. Il y a en gros deux types de vectorisation : implicite et explicite.

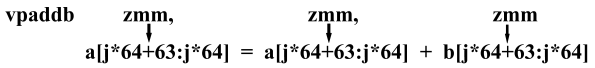


FIGURE 2.2 – Schéma d'exécution de l'instruction vectorielle vpaddd zmm, zmm, zmm.

2.1.1.2.1 La vectorisation implicite

On peut distinguer dans la vectorisation implicite l'auto-vectorisation par le compilateur et l'utilisation de librairies ou d'API. Ainsi les compilateurs ICC d'Intel® (Wang *et al.*, 2009), GCC (Edelsohn *et al.*, 2005) et LLVM (Clang, 2020) ont des capacités d'auto-vectorisation. Les librairies ou les API sont quant à elles des outils qui masquent la complexité de l'architecture SIMD aux programmeurs. Parmi ces API et librairies, on peut citer Sierra (Leissa *et al.*, 2014), ISPC, CUBA et OpenMP (Lee *et al.*, 2017), OpenCL, OpenACC et gSIMD (Wang *et al.*, 2014), Array Notation (Krzikalla et Zitzlsberger, 2016), Cyme (Ewart *et al.*, 2014), Vc (Kretz et Lindenstruth, 2012), Boost.SIMD (Estérie *et al.*, 2014), Neat SIMD (Gross, 2016), UME : :SIMD (Karpiński et McDonald, 2017), etc. Par exemple, avec OpenMP en environnement C/C++ supportant les *intrinsics* de l'architecture AVX-512 du fabricant Intel®, supposons que nous avons une fonction scalaire foo contenant une boucle for.

```
foo(){
     ...
     for(i=0; i<N; i++){
        a[i]=a[i]+b[i];
     }
     ...
}
```

Pour vectoriser cette boucle, nous pouvons utiliser la directive pragma de OpenMP

de la façon suivante :

```
#include <x86intrin.h> //appel du compilateur vectoriel
#include <omp.h> // inclusion de la librairie openMP
foo(){
     ...
     #pragma omp simd  //appel de la directive OpenMP
     for(i=0; i<N; i++){
        a[i]=a[i]+b[i];
     }
     ...
}
```

Pour une demande de parallélisation (architecture MIMD) on remplacerait simplement `#pragma omp simd` par `#pragma omp parallel for` et pour une demande de parallélisation et de vectorisation par `#pragma omp parallel for simd`. Si nous supposons dans cet exemple que a et b sont des tableaux d'entiers codés sur 8 bits chacun, c'est-à-dire que pour i donné $a[i], b[i]$ est un mot entier de 8 bits ; et que N est un multiple de 64, c'est à dire qu'il existe un nombre entier naturel strictement positif k tel que $N = 64k$, alors l'appel à la vectorisation par #pragma omp simd aura pour effet de remplacer l'addition scalaire à la compilation par l'instruction vectorielle *vpaddb zmm, zmm, zmm* qui sera exécutée k fois selon le schéma de la Figure 2.2. Dans ce schéma $a[i+p : i]$ représente la plage des valeurs allant de l'indice $a[i + p]$ à l'indice $a[i]$, c'est-à-dire selon la notation vectorielle $(a[i + p], \ldots, a[i])$. On voit effectivement dans cet exemple que, pour i fixé nul ou multiple de 64, au lieu de faire consécutivement la somme d'un élément $b[j]$ du tableau b (que nous assimilons à un vecteur b) où j prend tour à tour la valeur $i+63, \ldots, i+1, i$ et d'un élément $a[j]$ du tableau a (que nous assimilons à un vecteur a) où j prend tour à tour la valeur $i+63, \ldots, i+1, i$; on va d'un seul coup

sommer chaque $b[j]$ avec l'élément $a[j]$ correspondant. En théorie, l'exécution de l'instruction correspondant à cette boucle for se fait 64 fois plus vite dans ce cas.

2.1.1.2.2 La vectorisation explicite

Pour ce qui est de la vectorisation explicite, il y a deux possibilités : écrire du code assembleur (à l'aide des instructions comme *vpaddb* sur la Figure 2.2) ou s'appuyer sur les *intrinsics* (les fonctions vectorielles) qui sont des opérateurs comparables du point de vue syntaxique et d'aisance dans leur manipulation pour qui comprend leur sémantique, à ceux de langages de programmation de haut niveau comme le C/C++. Ces opérateurs sont dits vectoriels parce qu'ils permettent, à la différence d'un opérateur scalaire, de traiter plusieurs mots à la fois. Par exemple, sur une architecture SIMD avec des registres de 512 bits, supposons qu'on ait deux vecteurs v_1 et v_2 d'entiers codés sur 8 bits chacun. La dimension de chaque vecteur est alors de 64 (chaque champ étant un mot entier). Formellement, si nous notons $\oplus$ l'opérateur d'addition vectorielle, $+$ l'opérateur d'addition scalaire ; $v_1 = (x_0^1, x_1^1, \ldots, x_{63}^1), v_2 = (x_0^2, x_1^2, \ldots, x_{63}^2)$ et $v = v_1 \oplus v_2$ on a $v_i = (x_j^i), j \in \{0, \ldots, 63\}$ et $v = (x_k), k \in \{0, \ldots, 63\}$ avec $x_k = x_k^1 + x_k^2$. Ainsi, en une seule opération vectorielle (avec l'opérateur $\oplus$), les 64 opérations scalaires (avec l'opérateur $+$) correspondant à l'équation $x_k = x_k^1 + x_k^2$ sont effectuées alors qu'il faut exécuter une opération scalaire 64 fois pour obtenir le même résultat sur une architecture non vectorielle. Cet équivalent vectoriel est également plus écologique, car elle consomme beaucoup moins d'énergie que la suite d'instructions scalaires nécessaires pour effectuer la même opération (Steigerwald et Agrawal, 2011). Dans le cas d'Intel®, pour un processeur supportant l'AVX512BW, cet exemple est mis en œuvre à travers l'*intrinsic* $\_mm512\_add\_epi8$ qui est traduit en son équivalent en assembleur *vpaddb*. En effet, les fonctions vectorielles (intrinssics) ont leur équivalent direct en assembleur et ont donc, en termes d'ef-

ficacité et d'efficience, les mêmes valeurs que du code assembleur (Schmidt *et al.*, 2018). Toutes les études de benchmark à notre connaissance montrent que de façon générale la vectorisation explicite est plus efficiente en consommation d'énergie et supérieure en performance à la vectorisation implicite avec un facteur pouvant aller jusqu'à 8 (Amiri et Shahbahrami, 2020; Pohl *et al.*, 2016; Mitra *et al.*, 2013). Si l'on ajoute accessoirement à cela le fait que le développement des librairies et des API et leur maintenance requièrent une compréhension de codes développés en s'appuyant sur la vectorisation explicite, alors cette compréhension, qui suppose elle-même de comprendre les fonctions vectorielles ou leurs équivalents en assembleur, devient incontournable pour la programmation vectorielle. Mais coder en assembleur exige en plus de gérer soi-même des opérations de très bas niveau comme l'allocation des registres et le contrôle des types de données (Mitra *et al.*, 2013). Donc, finalement le meilleur rapport bénéfice/coût revient à adopter l'approche de programmation reposant sur les fonctions vectorielles ou *intrinsics*. Or comprendre la sémantique des fonctions vectorielles n'est pas aisée, et il en est donc de même de codes produits avec ces fonctions.

2.1.2 L'aide à la programmation parallèle

La programmation parallèle, malgré les avantages en termes de performance, d'économie et d'écologie qu'elle présente, est un paradigme de programmation autrement plus ardu que le paradigme de programmation scalaire séquentiel (Quinn, 2003). Aussi plusieurs stratégies ont été développées pour aider les programmeurs à l'adopter et à l'utiliser. Parmi ces stratégies, celle consistant en la recherche, le développement et le déploiement d'outils de visualisation est l'une des plus utilisées. Elle a contribué, au-delà du génie logiciel, à enrichir une discipline qui est la visualisation de logiciel en abrégé SV pour *software visualization* (en anglais). La SV est utilisée depuis longtemps dans la programmation de façon générale et

Synopsis

```
__m256i _mm256_maskz_unpacklo_epi32 (__mmask8 k, __m256i a, __m256i b)
#include <immintrin.h>
Instruction: vpunpckldq ymm {z}, ymm, ymm
CPUID Flags: AVX512F + AVX512VL
```

Description

Unpack and interleave 32-bit integers from the low half of each 128-bit lane in `a` and `b`, and store the results in `dst` using zeromask `k` (elements are zeroed out when the corresponding mask bit is not set).

Operation

```
DEFINE INTERLEAVE_DWORDS(src1[127:0], src2[127:0]) {
        dst[31:0] := src1[31:0]
        dst[63:32] := src2[31:0]
        dst[95:64] := src1[63:32]
        dst[127:96] := src2[63:32]
        RETURN dst[127:0]
}
tmp_dst[127:0] := INTERLEAVE_DWORDS(a[127:0], b[127:0])
tmp_dst[255:128] := INTERLEAVE_DWORDS(a[255:128], b[255:128])
FOR j := 0 to 7
        i := j*32
        IF k[j]
                dst[i+31:i] := tmp_dst[i+31:i]
        ELSE
                dst[i+31:i] := 0
        FI
ENDFOR
dst[MAX:256] := 0
```

FIGURE 2.3 – Description textuelle de l'*intrinsic* _mm256_maskz_unpacklo_epi32 telle que disponible sur `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#text=_mm256_maskz_unpacklo_epi32&ig_expand=7101`, le site d'Intel®.

dans la programmation parallèle (MIMD) en particulier. Mais à notre connaissance, elle n'a pas encore été appliquée à la compréhension de comportement de programmes vectoriels et c'est là l'une des originalités de notre contribution.

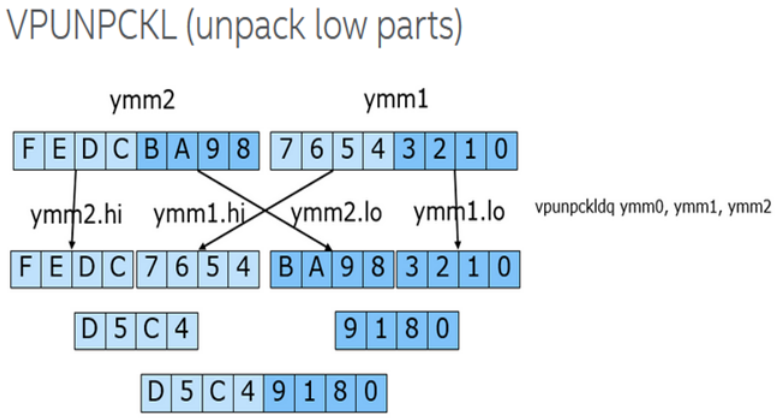


FIGURE 2.4 – Description graphique de l'instrinsic _mm256_maskz_unpacklo_epi32 proposée par un ingénieur d'Intel® et disponible sur le blog sur `https://www.isus.jp/specials/programming-using-avx2-permutations/`.

2.1.2.1 Exemples d'utilisation de la SV dans la programmation parallèle (MIMD) et illustration du besoin de visualisation en programmation vectorielle (SIMD)

Parmi les exemples d'utilisation de la SV, on peut mentionner PUMA-V, destiné à abaisser la barrière à l'entrée dans l'optimisation de codes sources afin de favoriser une auto-parallélisation par le compilateur (Papenhausen *et al.*, 2016). En effet pour tirer le meilleur parti de cette possibilité d'auto-parallélisation, l'utilisateur-développeur doit fixer lui-même certains paramètres afin d'avoir un code source optimal du point de vue de l'exécution parallèle au sortir du compilateur (Papenhausen *et al.*, 2016). Dans le même ordre d'idée, a été également développé Parceive, un outil pour comprendre le comportement d'un programme parallèle à différents niveaux d'abstraction, mais aussi identifier les possibilités de parallélisation que peut offrir un programme et le restructurer en conséquence (Wilhelm *et al.*, 2016). Ravel quant à lui, est un outil de visualisation développé pour analy-

```
__m256i enc_reshuffle(__m256i input) {
  __m256i in = _mm256_shuffle_epi8(input, _mm256_set_epi8(
        10, 11,  9, 10,  7,  8,  6,  7, 4,  5,  3,  4, 1,  2,  0,  1,
        14, 15, 13, 14, 11, 12, 10, 11, 8,  9,  7,  8, 5,  6,  4,  5
    ));
  __m256i t0 = _mm256_and_si256(in, _mm256_set1_epi32(0x0fc0fc00));
  __m256i t1 = _mm256_mulhi_epu16(t0, _mm256_set1_epi32(0x04000040));
  __m256i t2 = _mm256_and_si256(in, _mm256_set1_epi32(0x003f03f0));
  __m256i t3 = _mm256_mullo_epi16(t2, _mm256_set1_epi32(0x01000010));
  return _mm256_or_si256(t1, t3);
}
```

FIGURE 2.5 – Fonction enc_reshuffle (Muła et Lemire, 2018).

— We initially have four 6-bit values stored in each of the four bytes, with zeros elsewhere $[00d_5d_4d_3d_2d_1d_0|00c_5c_4c_3c_2c_1c_0|00b_5b_4b_3b_2b_1b_0|00a_5a_4a_3a_2a_1a_0]$.
— We call the `vpmaddubsw` (`_mm256_maddubs_epi16`) instruction providing as inputs our data as well as the vector made of the byte values `0x01, 0x40, 0x01, 0x40, ...`. This instruction multiplies pairs of 8-bit integers from its two input vectors, producing intermediate 16-bit integers, it then adds adjacent pairs of 16-bit integers (the first two, the third and the fourth, ...). In our case, we multiply alternatively by `0x40` (a shift by 6 bits) and `0x01` (the identity). By inspection, the output is $[0000c_5c_4c_3c_2|c_1c_0d_5d_4d_3d_2d_1d_0|0000a_5a_4a_3a_2|a_1a_0b_5b_4b_3b_2b_1b_0]$.
— We then call the `vpmaddwd` (`_mm256_madd_epi16`) instruction. It is similar to the `vpmaddubsw` instruction: it multiplies pairs of 16-bit integers from its two input vectors, producing intermediate 32-bit integers, it then adds adjacent pairs of 32-bit integers (the first two, the third and the fourth, ...). We call it with the vector made of the 16-bit values `0x0001, 0x1000, ...` so that we alternatively shift by 12 bits or compute the identity. By inspection, the result within each 32-bit word is $[00000000|a_5a_4a_3a_2a_1a_0b_5b_4|b_3b_2b_1b_0c_5c_4c_3c_2|c_1c_0d_5d_4d_3d_2d_1d_0]$.

FIGURE 2.6 – Explication textuelle du code de la fonction enc_reshuffle (Muła et Lemire, 2018).

ser et comprendre le comportement d'un code parallèle grâce à ses traces d'exécution et suivant un temps logique (par opposition au temps chronologique) (Isaacs *et al.*, 2014). Enfin, parmi ces quelques exemples que nous avons retenus, il a été montré à travers SeeMore que la visualisation peut permettre à une très large audience de comprendre les notions de base du parallélisme (Li *et al.*, 2017).

En ce qui concerne la programmation vectorielle, un certain nombre de faits soulignent le besoin et l'intérêt qu'il y aurait à utiliser cette technique pour assister les programmeurs. Le premier de ces faits découle de l'observation des fabricants de matériels. En effet, Intel® par exemple met à la disposition des développeurs une documentation fournie expliquant le fonctionnement des fonctions

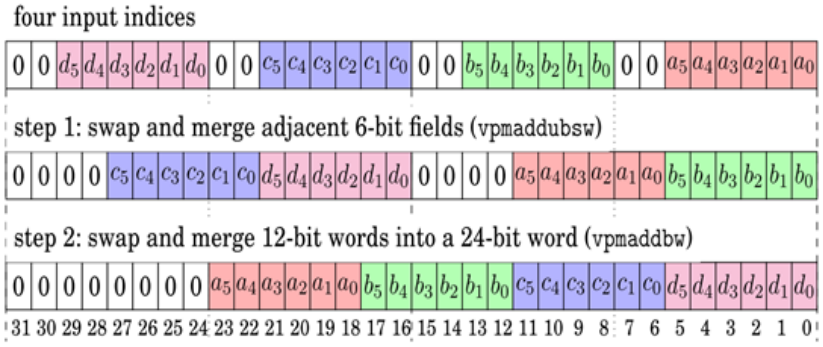


FIGURE 2.7 – Explication graphique du code de la fonction enc_reshuffle (Muła et Lemire, 2018).

vectorielles. On a accès à la description des fonctions vectorielles sous format XML sur le site d'Intel® et à une version interactive sur le même site. Les ingénieurs d'Intel® sont bien conscients des limites de telles documentations textuelles puisque ponctuellement dans leurs publications, pour des explications de la sémantique d'une fonction vectorielle précise, ils font recours à des illustrations sous forme d'images ou de graphiques. On peut par exemple saisir les différences d'effort mental à fournir pour comprendre le fonctionnement de la fonction vectorielle $\_mm256\_maskz\_unpacklo\_epi32$ à travers une description textuelle (Figure 2.3) de cette fonction disponible sur le site d'Intel® (Intel, 2018) et l'explication graphique (Figure 2.4) de la même fonction disponible sur un blog (VPUNPCK, 2018) entretenu par un ingénieur sur le même site. Le second fait découle de l'observation des développeurs d'applications vectorielles. Ils se rendent bien compte là aussi des difficultés et des limites de l'explication textuelle dans les codes qu'ils produisent. Ainsi, on peut voir la différence entre l'explication textuelle (Figure 2.6) du code de la fonction enc_reshuffle (Figure 2.5) et son explication graphique (Figure 2.7). Bien qu'elles soient des applications ponctuelles sur des exemples isolés des techniques de visualisation, les observations précédentes illustrent le potentiel de la SV appliquée à la programmation vec-

torielle. Toutefois, ces démarches présentent plusieurs limites. La première limite est que ces représentations visuelles ne s'appliquent bien évidemment qu'aux seuls cas particuliers auxquels s'intéressent leurs auteurs. Mais le plus gênant est que ces représentations visuelles sont susceptibles de ne pas aider d'autres personnes à comprendre la fonction vectorielle ou le code vectoriel visé, surtout si ces personnes sont des novices dans le domaine de la programmation vectorielle, cela à cause de l'effet *expert blind spot*. Cet effet est susceptible de masquer le besoin de comprendre du novice par le désir de l'expert de présenter ses connaissances, ce qui peut créer un biais pour le novice, car pour l'expert le souci n'est pas le contenu à expliquer, mais sa propre explication (Nathan *et al.*, 2001). Enfin, ces représentations visuelles relèvent davantage de l'intuition de leurs auteurs que d'une démarche rigoureuse mobilisant par exemple les acquis méthodologiques de la SV ou de l'InfoViz. Malgré toutes ces limites, ces représentations sont quand même de la visualisation.

2.2 Cadre cognitif

De façon générale, la visualisation est une perception dont le traitement mental sollicite la mémoire visuelle par opposition à la mémoire symbolique (Luck et Hollingworth, 2008). C'est l'activité cognitive qui consiste à former une image mentale d'une perception, ou même à «former un modèle mental de quelque chose» (Spence, 2014). Ainsi, la visualisation n'a *a priori* rien à voir avec l'ordinateur (ou l'informatique). Reste que le processus cognitif que constitue la visualisation peut être facilité par des représentations visuelles externes (Mazza, 2009). En clair, un langage visuel (ou iconique) peut amplifier les performances de la visualisation en tant que processus de traitement de l'information du point de vue de la cognition humaine. C'est peut-être pourquoi certains (van Wijk, 2006; Munzner, 2014) assimilent la visualisation à une perception consécutive à un sti-

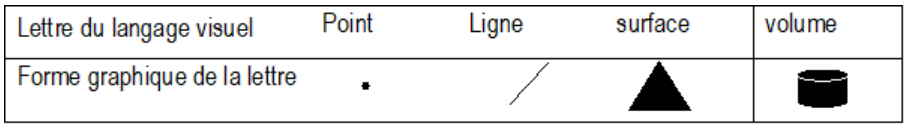


FIGURE 2.8 – Lettres du langage visuel selon Bertin (1983).

mulus visuel et même aux technologies et méthodes permettant de produire de tels stimuli.

2.2.1 Le langage visuel

Définir ce qu'est le langage visuel n'est pas aisé. En fait l'expression langage visuel signifie différentes choses pour différentes personnes (Erwig *et al.*, 2017). Rien que du point de vue de la programmation, un langage visuel peut signifier un langage qui permet de programmer en utilisant des pictogrammes, mais il peut aussi signifier un langage qui permet de comprendre la structure, l'évolution ou le comportement d'un programme déjà créé (Zhang, 2010). Nous adopterons une définition plus cognitive, en liant le langage visuel à la visualisation en tant que processus cognitif. Ainsi, un langage visuel est un langage dont le traitement du point de vue cognitif fait appel à la mémoire visuelle. Cette définition a pour caractéristique principale d'intégrer la cognition (humaine) dans la boucle. Nous adoptons ainsi une perspective pratique, le souci étant de situer un langage visuel par rapport à un langage non visuel (un langage symbolique comme ce texte que vous lisez par exemple). Les variables d'un tel langage ont été formalisées ; les lettres y sont des points, des lignes, des surfaces ou des volumes (Figure 2.8) (Bertin, 1983). Pour former un mot avec ces lettres, on identifie sept variables possibles associées à ces lettres (Figure 2.9) (Bertin, 1983) :

— la position dans l'espace, il s'agit plus précisément de l'espace plan à deux dimensions ;

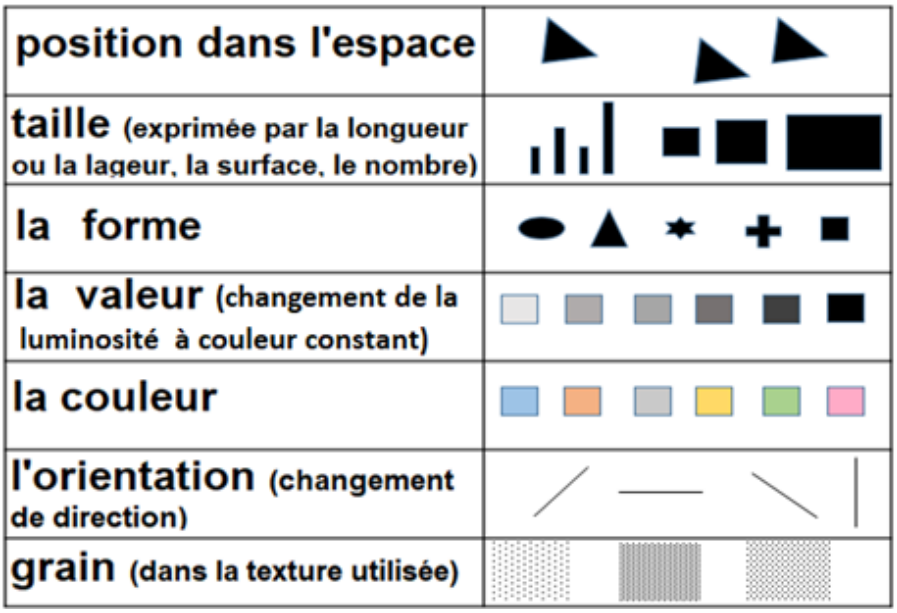


FIGURE 2.9 – Mots du langage visuel (Bertin, 1983).

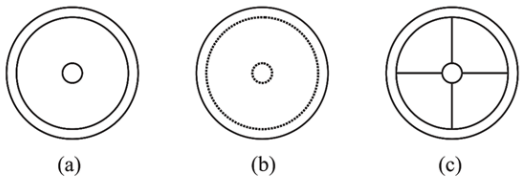


FIGURE 2.10 – Illustration de la règle des regroupements visuels (Diehl, 2007).

— la taille (exprimée par la longueur ou la largeur, la surface ou le nombre) ;
— la forme géométrique ;
— la valeur de la luminosité ou saturation de la teinte (couleur) ;
— la couleur ;
— l'orientation ou changement de direction ;
— et enfin le grain dans la texture utilisée.

Les phrases visuelles, qui sont en fait des *patterns*, sont obtenues suivant des critères de regroupement dont les principaux sont par ordre décroissant de priorité : connectivité, proximité, similitude de couleur ou de forme, continuité des courbes,

symétrie, fermeture des zones, taille relative, orientation, arrière-plan, et transparence. On peut par exemple remarquer que sur la Figure 2.10 le petit cercle en (c) forme un même groupe d'objets avec le cercle auquel il est relié, en (a) on voit plutôt une couronne avec un petit cercle au milieu et enfin en (b) on a encore comme en (a) regroupé le petit cercle et le grand cercle intérieur. Ces règles s'appuient sur les lois de Gestalt (Wertheimer, 1938, 2012; Palmer et Rock, 1994) et nous utilisons par la suite particulièrement les quatre premières, notamment pour décrire des relations de similitude, de proximité ou fonctionnelles. Suivant ces règles, il s'agit de manipuler les images de sorte que les relations entre elles soient mises en exergues et perçues, c'est-à-dire que des *patterns* apparaissent (Bertin, 1983). Les atouts du langage visuel reposent sur plusieurs facteurs.

2.2.2 L'efficacité du langage visuel et la fonction cognitive des métaphores

La mémoire visuelle est beaucoup plus efficace et efficiente en termes de mémorisation et même de rappel que la mémoire symbolique (Standing, 1973; Palmeri et Tarr, 2008; Schurgin, 2018; Baddeley et Lieberman, 2017). Outre cette performance en matière de stratégie dans le traitement de l'information, d'autres facteurs intrinsèques au langage visuel lui confèrent ses avantages (Petre et Blackwell, 1998). Il permet ainsi :

- De condenser l'information en dissipant le bruit ;
- D'avoir un niveau d'abstraction plus élevé ;
- De faciliter la compréhension tout en étant plus attrayant ;
- De communiquer de façon moins formelle (donc de façon plus riche) que le langage symbolique.

Dans le même ordre d'idée, le langage visuel est plus adapté aux métaphores et réciproquement si l'on considère avec Lakoff et Johnson (2008) le fait que les métaphores sont inhérentes à la nature même de la cognition humaine et à son

fonctionnement, et non de simples figures de style linguistiques. En effet, le signe symbolique, c'est-à-dire la langue, est susceptible d'après Peirce et Bucher (1955) de passer par le signe iconique pour être appréhendé. Le langage visuel, d'essence iconique, fait l'économie de cette étape supplémentaire. On peut donc comprendre l'importance des métaphores dans la visualisation de logiciel.

Plus généralement, en remarquant que le langage symbolique est l'outil qui nous permet à la fois nommer les choses (usage en contexte direct), mais aussi d'exprimer nos idées ou de réfléchir sur les choses (usage en contexte indirect) (Carnap, 1997), il se donne comme moyen d'expression de notre représentation du monde dont nous sommes constitutifs. Étant donné que le monde préexiste à sa représentation, et que notre rapport au monde est antérieur au langage, le langage n'est pas le premier moyen d'expression de notre représentation du monde, car nous savons ou plutôt nous sentons des choses du monde plus tôt et plus souvent que nous ne les exprimons ou même que nous ne pouvons les exprimer. De ce point de vue, la thèse de Lakoff et Johnson (2008) acquiert une importance fondamentale dans l'explication de l'efficience et de l'efficacité du langage visuel ou iconique par rapport au langage symbolique. En effet, la métaphore n'est plus seulement un mécanisme de la cognition mise en œuvre dans le langage symbolique, mais le langage symbolique devient une métaphore nous permettant d'exprimer notre rapport au monde ; en clair notre rapport au monde serait d'emblée donné dans un langage non symbolique, mais visuel ou apparenté ; c'est-à-dire un langage moins abstrait. L'efficacité et l'efficience du langage visuel viennent donc, sous cette perspective, de ce qu'il permet de faire l'économie du processus d'abstraction que constitue la mise en œuvre de la métaphore nécessaire au passage au langage symbolique.

2.3 La visualisation de logiciel : méthodes, moyens et problèmes

Plusieurs techniques de visualisation ont bâti sur ces atouts du langage visuel pour aider dans l'exécution des tâches mentales. C'est ce qu'on appelle l' amplification de l'intelligence (AI) par opposition à l' intelligence artificielle (IA) (Brooks Jr, 1996). Cette distinction est très importante, car certaines données et certaines tâches doivent faire l'objet d'automatisation IA plutôt que de visualisation (AI). Cette séparation, le chercheur en visualisation, doit être en mesure de l'opérer afin de résoudre le bon problème avec le bon outil (Sedlmair *et al.*, 2012).

2.3.1 La visualisation de logiciel

On regroupe les techniques de visualisation selon les domaines d'application. On a ainsi la visualisation de données (DataViz)—pour *Data Visualization* en anglais—et la visualisation d'information ou InfoViz. La différence entre les deux vient essentiellement du type de données traitées par la visualisation. En visualisation de l'information, les données sont beaucoup plus abstraites qu'en visualisation de données. Cependant, il est difficile de tracer une limite claire entre les deux domaines (Kim *et al.*, 2016; Nagel, 2006). La visualisation de logiciel ou SV (pour *Software Visualization* en anglais) fait partie de l'InfoViz (Diehl, 2007). Myers (1986) a grandement contribué à fixer les concepts de la SV et a établi une taxonomie du domaine qu'il n'a cessé d'affiner (Myers, 1990). D'après cette taxonomie, la visualisation de programme est l'utilisation d'images graphiques pour illustrer un programme, et ce après sa création. Cette définition permet de distinguer la visualisation de programme (*program visualization* (PV) en anglais) de la programmation visuelle (*visual programming* (VP) en anglais). Dans la VP, les images graphiques sont aussi utilisées, mais pour créer le programme. On peut comprendre dès lors qu'il y ait pour la VP le besoin de développer des langages de

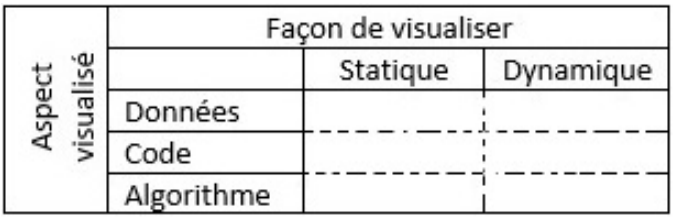


FIGURE 2.11 – Taxonomie de la visualisation de programme (Myers, 1990).

programmation visuels, donc basés sur des grammaires formelles visuelles ; alors que pour la PV il est davantage question de trouver les métaphores, en tout cas les images les plus évocatrices (Diehl, 2007). Il faut souligner aussi que, le coloriage ou l'indentation de code, même sans le support des images, sont déjà de la PV (Myers, 1986). La taxonomie de la visualisation de programme apparaît sous la forme d'un tableau à double entrée (Figure 2.11). Les colonnes de ce tableau représentent de quelle façon la visualisation d'un programme peut être effectuée. Cette visualisation peut être en effet statique ou dynamique. Elle est statique si elle consiste en des images fixes reflétant l'exécution ou le comportement du programme à des points particuliers. Elle est dynamique si elle consiste en une animation illustrant l'exécution ou le comportement du programme. Les lignes du tableau représentent quant à elles quel aspect d'un programme peut être visualisé. On peut ainsi :

— visualiser les données issues de l'exécution d'un programme ;

— visualiser le code source d'un programme. Un exemple de cette forme de visualisation est l'indentation ou la coloration automatiques de codes dans certains environnements de programmation ;

— visualiser l'algorithme qu'un programme exécute. La visualisation de l'algorithme d'un programme diffère beaucoup des deux précédentes formes de visualisation en ce sens qu'il convient d'abord d'abstraire l'information à visualiser. De cette façon, ce qui est visualisé peut ne pas correspondre

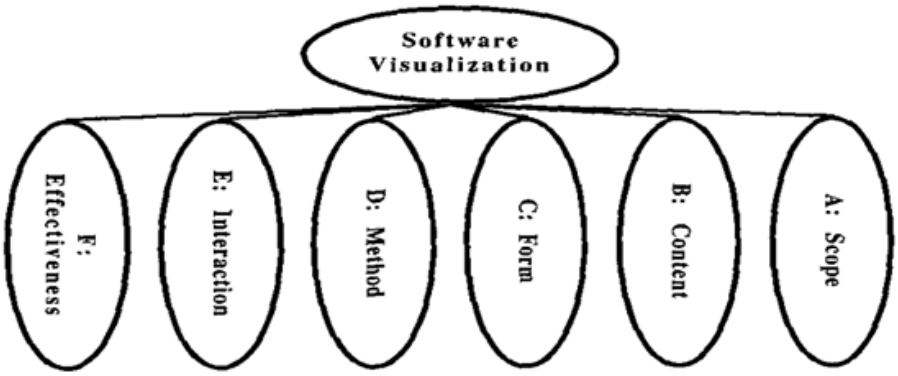


FIGURE 2.12 – Taxonomie de la visualisation de logiciel (niveau I) (Price *et al.*, 1993).

à un bout spécifique du programme, mais être plutôt l'illustration d'un comportement correspondant à un niveau d'abstraction plus élevé.

À la suite de Myers, la PV a été renommée en SV. Telle qu'utilisée aujourd'hui, la PV est incluse dans la SV. La PV recouvre la visualisation de code et de données de programmes. La SV recouvre en plus de la PV, la visualisation des algorithmes et même la visualisation d'autres aspects des programmes comme leur évolution au fil du temps, leur performance durant l'exécution, etc. (Price *et al.*, 1998). Une taxonomie plus globale a été proposée (Price *et al.*, 1993). Cette taxonomie (Figure 2.12) a plusieurs niveaux de profondeur et est évolutive en plus d'intégrer celle de Myers. Ce besoin d'évolutivité est nécessaire, la visualisation de logiciel étant un domaine de recherche encore très actif. Cette nouvelle taxonomie a également l'avantage de relier la visualisation de logiciel à la pratique plus générale du génie logiciel. Elle est plus systématique. Nous la retenons comme taxonomie de base, car toutes les autres, que ce soit celle de Myers (1990) (Figure 2.11) ou d'autres (Diehl, 2007; Chi, 2000), peuvent y être intégrées, l'inverse n'étant pas toujours possible. Cette taxonomie prend en compte le point de vue des acteurs intervenants dans la visualisation de logiciel :

— Du point de vue du programmeur qui écrit l'algorithme ou développe le

système à visualiser, ce qui est intéressant c'est la portée du système de visualisation. Cette portée se décompose dans le cadre de cette taxonomie en deux sous-catégories :

i) la généralité du système qui traduit sa capacité à permettre la visualisation d'un ensemble générique et extensible de programmes ou *a contrario* seulement d'un groupe de programmes fixés dès la conception du système.

ii) l'évolutivité qui est la capacité du système à permettre la visualisation des programmes très grands en termes de complexité ou de lignes de codes.

— Du point de vue du développeur du système de visualisation, ce qui est intéressant c'est l'aspect du programme à visualiser. C'est-à-dire, s'agit-il de code, de données, d'algorithmes ou d'un autre aspect du programme comme sa performance ou son évolution dans le temps en termes de maintenance par exemple. On peut noter que ce point de vue intègre toutes les lignes de la taxonomie de Myers. Selon l'aspect visualisé, la métaphore utilisée correspond à un certain niveau d'abstraction. Ainsi, le plus bas niveau d'abstraction est celui du code, et l'algorithme fait partie des niveaux d'abstraction plus élevés. Enfin, pour le développeur, il est important de savoir comment, et quand les données à visualiser sont recueillies, et si elles sont exhaustives.

— Un autre point de vue est celui de la caractérisation du système de visualisation par le résultat en sortie. C'est-à-dire à quel type d'environnement physique et technique ce résultat est-il destiné et avec quelle granularité ? S'agit-il d'un écran ou d'un environnement virtuel ? Quel est le style de présentation adopté, s'agit-il simplement de graphisme ou y a-t-il des animations et même du son ? Jusqu'à quel point le système de visua-

lisation permet-il de visualiser de manière synchronisée différents aspects d'un même système ? Permet-il de visualiser différents systèmes de manière synchronisée ?

— Du point de vue du concepteur qui spécifie le système de visualisation, il importe de savoir s'il doit être développé de bout en bout ou si tout ou partie d'un système ou librairie existant doit être réutilisé. Il importe également de savoir comment le lien est assuré entre le système de visualisation et le système visualisé ; faut-il par exemple, instrumenter ou annoter le système visualisé ?

— Du point de vue de l'utilisateur final du système de visualisation, ce qui importe c'est la convivialité du système, comment il le contrôle et interagit avec lui. Offre-t-il par exemple la possibilité de gérer de façon interactive une visualisation en cours, de marquer une pause ou un retour en arrière à un point précis choisi ?

— Toujours du point de vue de l'utilisateur, un autre élément important est la qualité de l'information communiquée par le système de visualisation. Cette information est-elle claire et permet-elle de comprendre réellement le comportement du système visualisé ? Jusqu'à quel point le système a-t-il fait l'objet d'une bonne évaluation expérimentale ? Enfin, le système est-il en production ou est-il seulement un prototype ?

Diehl (2007) a essayé de fixer un cadre à la visualisation de logiciel et de faire le point mis à jour à cette date-là. Il définit la SV comme la visualisation des artefacts liés au logiciel et à son processus de développement. Cette visualisation peut porter ainsi sur la structure, le comportement ou l'évolution du logiciel. De ces trois aspects, il convient de noter que notre démarche s'inscrit dans la visualisation du comportement. C'est l'aspect pour lequel il existe le moins d'outils produits (Chotisarn *et al.*, 2020). Le but de la visualisation de logiciel est de créer des images mentales permettant une meilleure compréhension du système à visualiser. À cet

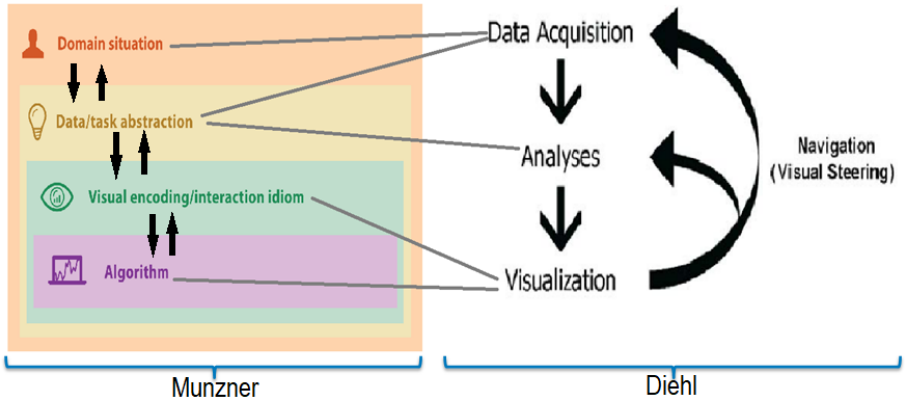


FIGURE 2.13 – Reconciliation des démarches de la visualisation de l'information (InfoViz) (Munzner, 2014) et de la visualisation de logiciel (SV) (Diehl, 2007).

effet, l'aspect évocatoire de la visualisation de logiciel est très important (Diehl, 2007). Dans ce processus évocatoire, l'emphase est mise sur l'importance des métaphores bien choisies. Que ce soient les propriétés statiques ou dynamiques d'un logiciel, il est toujours assez difficile de trouver les métaphores adéquates permettant de rendre compte correctement de ces propriétés. Les aspects statiques sont ceux qui peuvent être déterminés avant l'exécution du programme. Les propriétés dynamiques ne peuvent être calculées que pendant l'exécution. Pour ces propriétés dynamiques, il y a des écueils évidents liés au recueil des données y afférentes. En effet, un risque notable dans ce recueil est que certaines techniques utilisées, comme l'instrumentation, modifient le comportement du système à visualiser et peut donc biaiser la compréhension de ce comportement (Diehl, 2007).

2.3.2 Méthodes et moyens de la SV

Diehl (2007) propose également une démarche pour aborder tout problème de visualisation : le pipeline de la visualisation. Cette démarche s'articule en trois grandes étapes (partie droite de la Figure 2.13) :

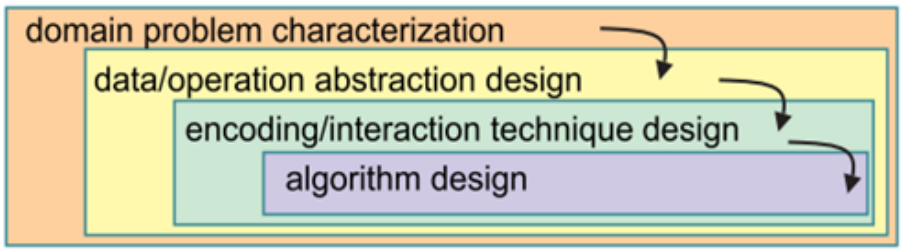


FIGURE 2.14 – Demarche méthodologique emboîtée de la visualisation de l'information (Munzner, 2009).

i) L'acquisition des données. Il s'agit de se procurer les données à utiliser pour construire la visualisation. Leur nature dépend de l'aspect que l'on veut visualiser. Ainsi, pour visualiser la structure d'un système, il s'agit de code source, pour visualiser son comportement, il s'agit de données recueillies pendant l'exécution, et pour visualiser l'évolution il s'agit de données de suivi de versions.

ii) Après cette étape d'acquisition des données, il convient de les analyser, y compris par des méthodes statistiques afin de les réduire quantitativement et de les améliorer qualitativement.

iii) Les données issues de l'analyse sont mises en correspondance avec le modèle visuel, c'est-à-dire transformées en images et formes géométriques appropriées pour être affichées sur un écran de manière statique ou animée ou même rendues en réalité virtuelle. Comme on peut le remarquer, chaque étape utilise les résultats de l'étape qui la précède.

Il faut néanmoins reconnaître que c'est Munzner (Figure 2.14) qui a proposé une démarche rigoureuse, opérationnelle et complète dans tout projet de visualisation de l'information (Munzner, 2009). Il s'agit d'une démarche emboîtée en quatre étapes où comme dans le modèle de Diehl (2007) qu'il systématise et complète, une étape utilise les données de l'étape précédente. Ces quatre étapes sont :

i) La caractérisation du domaine du point de vue des besoins de l'utilisateur final du système ;

ii) La description du domaine tel que caractérisé à l'étape précédente par des données concrètes. Il s'agit d'abstraire les opérations en relation avec les caractéristiques intéressantes du domaine et de retenir les données liées à ces opérations, soit que ces données en sont issues, soit qu'elles les décrivent. Ces données doivent, si besoin est, être transformées en données utilisables pour la visualisation ;

iii) L'encodage des données issues de l'étape précédente sous forme graphique et la conception de leur interaction. Idéalement, cet encodage doit reposer sur des fondements cognitifs ;

iv) La conception et l'implémentation des algorithmes pour permettre la visualisation effective des résultats de l'étape précédente sur un écran, en réalité virtuelle, etc. Cette étape relève du génie logiciel dont on doit mettre à profit les acquis et la méthodologie.

L'un des plus importants aspects de cette démarche est qu'elle formule des propositions pour une double validation de chacune de ses étapes (Figure 2.15). La première validation se fait à étape constante (validation immédiate) et la deuxième validation se fait après la validation de l'étape qui suit. On voit que de cette façon, non seulement le système est emboîté, mais on a un système de validations imbriquées où les validations des étapes les plus extérieures encadrent les validations des étapes les plus intérieures. Cela réduit le risque de se tromper sur une étape en amont et de propager cette erreur sur les étapes en aval, mais en même temps toute la démarche est validée par l'extrant final. Pour chacune de ces étapes, la démarche permet d'identifier les risques auxquels il faut faire face en répondant à un certain nombre de questions.

— Pour les risques liés au domaine, il faut bien caractériser le domaine du

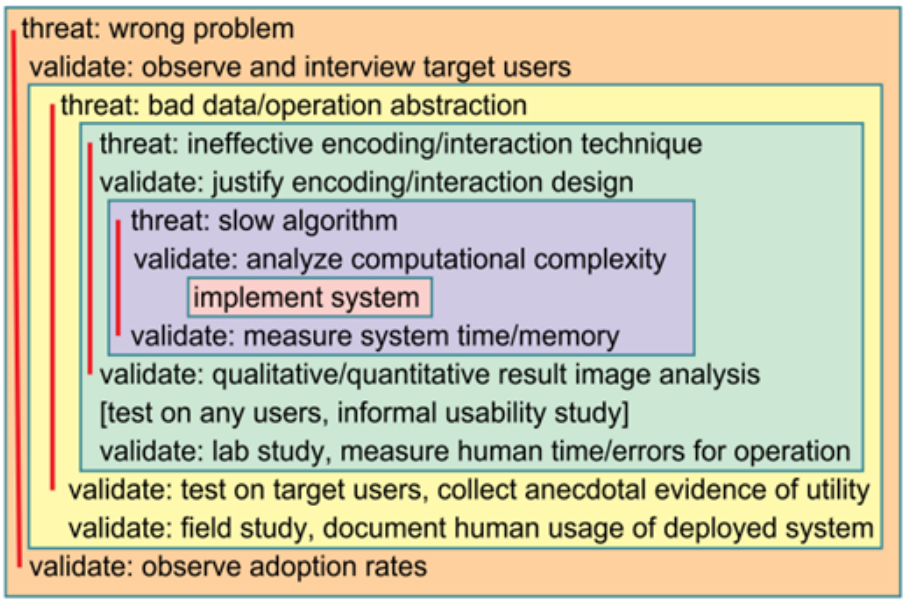


FIGURE 2.15 – Identification des risques, validations immédiates et imbriquées de chaque étape de la démarche (Munzner, 2009).

point de vue des besoins réels des utilisateurs du système. Cela peut se faire dans l'immédiat par simple observation de ces utilisateurs ou même en les interviewant. En aval, cela se fait en observant le taux d'adoption ou d'acceptation de l'outil implémenté même s'il y a là un risque de faux positifs.

— Pour les risques liés aux données à retenir pour décrire le domaine tel que caractérisé à l'étape précédente, il faut vérifier si ces données permettent aux utilisateurs de résoudre leur problème. Une façon de le faire est de tester un prototype avec ces utilisateurs, avant l'implémentation proprement dite. Non maîtrisés, ces risques peuvent avoir un impact négatif sur l'utilité du système.

— Pour les risques liés à l'encodage et à la conception des interactions, il faut vérifier si effectivement les images et l'animation éventuelle qui est faite de ces images améliorent et facilitent la compréhension du système

visualisé. Ces risques peuvent avoir un impact négatif surtout sur l'utilité du système s'ils ne sont pas maîtrisés. Pour maîtriser ces risques, il est possible de s'appuyer sur des fondements cognitifs.

— Enfin, pour les risques liés à l'implémentation, il faut s'appuyer sur les outils du génie logiciel afin de développer un système capable de gérer par exemple une montée en charge. Non maîtrisés, ils peuvent avoir un impact négatif sur l'utilisabilité du système.

Nous montrons sur la Figure 2.13 une correspondance entre les différentes étapes des démarches de Munzner (2009) et de Diehl (2007).

La démarche de Munzner (2009) laisse apparaître un projet de visualisation comme une suite de tâches pouvant être réalisées chacune de façon tout à fait indépendante. Chaque tâche correspond à une étape du modèle, et l'extrant de l'étape précédente constitue l'intrant de l'étape suivante. La validation dans l'immédiat se fait en vérifiant la consistance et la cohérence de l'extrant par rapport à l'étape suivante, et la seconde validation se fait plus tard après l'implémentation de l'étape suivante. Cette seconde validation intervient donc en principe après l'implémentation de l'outil de visualisation, mais des techniques comme celle du magicien d'Oz ou d'autres techniques de prototypage peuvent être utilisées pendant la phase de conception pour assurer cette validation Munzner (2009, 2014). Suivant cette perspective, Sedlmair *et al.* (2012) pensent qu'il est difficile qu'un même projet puisse apporter des contributions d'égale importance à toutes les étapes. Il convient dès lors, pour évaluer un projet de visualisation, de prendre en compte la contribution éventuelle correspondant à chacune des étapes, indépendamment du produit de l'étape finale que constitue l'outil de visualisation (Sedlmair *et al.*, 2012; Munzner, 2008, 2009, 2014). L'évaluation est justement un des problèmes récurrents soulevés dans le domaine de la visualisation de logiciel. Il s'agit de savoir comment garantir que l'évaluation du résultat à l'issue de la démarche, quelle que

soit cette démarche, soit valide et scientifiquement recevable. Merino *et al.* (2018) ont ainsi constaté que plus de 62% de projets dans le domaine de la visualisation de logiciel manquent d'évaluation rigoureuse et crédible. Ils ont en outre recensé les différentes techniques d'évaluation. Comme techniques très subjectives et peu rigoureuses d'évaluation, certains s'appuient sur des références théoriques pour affirmer l'effectivité de l'atteinte de leur objectif à travers l'outil de visualisation. D'autres encore s'appuient sur des évidences anecdotiques comme de constater qu'un ou quelques utilisateurs ont fait un feedback positif. Parmi les stratégies d'évaluation considérées comme rigoureuses, on peut citer plusieurs techniques :

i) Les scénarii de cas d'utilisation. Dans cette stratégie, des problèmes sont envisagés et des scénarii d'utilisation du système sont alors élaborés pour y répondre. Ordinairement, cette stratégie s'intitule exemple d'application, cas d'utilisation, exemple de cas, etc.

ii) Le sondage. On peut classifier les sondages à travers la façon de recueillir les données, mais aussi à travers les types d'analyse effectués sur ces données. Les données peuvent être recueillies par un questionnaire, par la méthode *think aloud*, par des interviews, par des enregistrements vidéo, ou de toute autre façon (les logs d'utilisation, les cartes émotives, des dispositifs de suivi en temps réel du mouvement des yeux, etc.). L'analyse quant à elle peut être quantitative ou qualitative.

iii) L'expérimentation. Dans cette technique, le chercheur peut suivre des variables dépendantes en manipulant des variables dont elles dépendent. Les utilisateurs sont alors affectés aléatoirement à l'utilisation du système. Quand ce procédé aléatoire n'est pas possible, on parle de quasi-expérimentation. Au sujet de l'expérimentation justement, Borgo *et al.* (2018) soulignent les opportunités et les risques que représentent actuellement la possibilité de recruter des participants en ligne pour évaluer et valider expérimentalement les

résultats des projets de recherche en visualisation d'information. En termes d'opportunité, la possibilité de recruter en ligne permet de surmonter les problèmes liés à la taille de l'échantillon et de la localisation géographique de l'échantillon : on peut recruter dans une population infiniment plus grande et géographiquement répartie dans les quatre coins du globe. En termes de risque, ce mode de recrutement exige d'adapter les outils de la collecte et limite les possibilités d'interaction : il faut donc s'assurer que les résultats restent valides par rapport à l'objectif visé et à la tâche assignée aux participants (Borgo *et al.*, 2018).

iv) L'étude de cas. C'est une technique qui consiste à décrire un contexte d'un projet dans lequel le système de visualisation a été utilisé pour aider les acteurs à surmonter un problème précis. Bien que le chercheur ait peu de contrôle sur l'étude comparée à l'expérimentation, elle permet d'analyser le système dans son environnement réel et en profondeur.

Peu importe la stratégie d'évaluation choisie, il faut préciser les propriétés à évaluer et expliciter la méthode d'évaluation pour pouvoir tirer des conclusions pertinentes et crédibles sur le système de visualisation évalué. Pour la validation des projets d'études de conception en InfoViz, Sedlmair *et al.* (2012) suggèrent les études de cas ou les scénarii d'utilisation. Certains auteurs, comme Seriai *et al.* (2014) regroupent tout simplement les deux méthodes de validation. Sensalire *et al.* (2009) pensent que dans certains cas le développeur de l'outil est le mieux placé pour l'évaluer et font plutôt une différence entre les types de participants (professionnels *vs* académiques ou chercheurs) plutôt qu'entre les méthodes d'évaluation. Au sens défini par Merino *et al.* (2018) nous pensons qu'une façon de distinguer entre étude de cas et scénario d'utilisation est le degré de découplage entre l'observateur de la visualisation et l'utilisateur de l'outil de visualisation. Dans l'étude de cas, l'utilisateur ne sait rien du processus de développement de l'outil et est distinct de l'observateur (en général le développeur) qui observe l'uti-

lisation et pose au besoin des questions à propos. Dans le scénario d'utilisation, l'utilisateur et l'observateur sont confondus.

2.3.3 Problèmes de la SV

Nous pouvons regrouper les problèmes rencontrer en SV en deux catégories : Ceux communs à l'InfoViz de façon générale, et ceux propres à la SV en ce qui a trait au comportement de code notamment, puisque l'évolution et la structure de code procédant de données qui sont données, c'est-à-dire disponibles ou réputées telles dès le début du processus de visualisation, peuvent plus facilement être assimilées aux autres domaines d'étude et d'application de l'InfoViz.

Pour ce qui est des problèmes communs de l'InfoViz donc, on peut d'abord remarquer qu'une science (ou un programme de recherche pour reprendre l'expression de Lakatos (1976)) consiste depuis Reichenbach (1938) d'une part à formuler des énoncés en veillant à s'assurer de leur cohérence (problème de la découverte) (Lakatos, 1970), et d'autre part à veiller à la correspondance de ces énoncés avec la réalité (problème de la justification) (Lakatos et Musgrave, 1969; Lakatos, 1980). L'InfoViz a des problèmes encore pendants relativement à ces deux aspects de la pratique scientifique. En effet, sont tout à fait d'actualité les discussions et les travaux aussi bien sur les fondements théoriques de l'InfoViz (Purchase *et al.*, 2008; Sedig et Parsons, 2016; Chen *et al.*, 2016; Correll, 2022) que sur sa validité observationnelle (Lorensen, 2004; Sedig et Parsons, 2016; van Wijk, 2006; Correll, 2022). Puisque nous nous inscrivons dans une perspective pratique, ce sont les discussions et les travaux autour de la correspondance à la réalité des énoncés de l'InfoViz en tant que science, c'est-à-dire sa validité observationnelle qui nous intéressent le plus. Mais il ne faut pas perdre de vue que les questions autour des fondements théoriques et de la validité observationnelle sont intimement et

intrinsèquement liées (Tardif, 2015; Robert, 2009; Meyer, 1979).

Pour ce qui est des problèmes propres à la SV, ils découlent de la nature même de son objet, de ce qu'on veut y visualiser, à savoir le comportement de code. Le texte d'un programme ou d'un algorithme ne renseigne pas sur son comportement puisqu'il faut procéder à une exécution abstraite (passer par un interpréteur humain par exemple) ou réelle (dans un environnement informatique) pour accéder à ce comportement. De ce fait, la transposition de certains acquis de l'InfoViz ne va plus de soi en ce qui concerne la SV et est susceptible de faire l'objet de travaux à part entière. Parmi ces acquis de l'InfoViz difficiles à transposer au domaine de la SV en ce qui a trait au comportement de code, on peut mentionner la technique consistant à analyser et transformer les données suivant les stades du processus de visualisation (Chi, 2000) ; la séparation entre référents et attributs qui est pourtant la norme pour d'autres domaines de la visualisation d'information (Purchase *et al.*, 2008). Dans d'autres domaines de l'InfoViz, après cette séparation, la visualisation consiste alors à rechercher des métaphores permettant de décrire les relations entre ces deux caractéristiques des données ou même simplement de représenter une des caractéristiques. La méthodologie de Munzner (2009), bien que beaucoup plus détaillée et opérationnelle que celle de Diehl (2007), doit également être adaptée. Parmi ces adaptations, la description du domaine et la caractérisation des données, qui correspondent à l'étape 2 (Figure 2.13) de la méthodologie de Munzner (2009), devient encore plus importante. Cette description et cette caractérisation s'opèrent à travers un modèle du domaine, qui dans le cas du comportement de code, permet de rendre compte de ce comportement à travers des données concrètes. Il faut souligner que du point de l'InfoViz, l'élaboration d'un tel modèle de domaine peut être en soi et à elle seule une contribution importante et suffisante dans le cadre de conférences des travaux ou même des journaux sur la visualisation par exemple (Munzner, 2008; Oppermann et Munzner, 2020). Ce

modèle, du point de vue de la SV et suivant la méthodologie de Munzner (2009), peut être validé de trois manières : soit en discutant et en le validant avec un expert du domaine, soit immédiatement dans le processus de mise en œuvre de la visualisation parce que l'encodage graphique que ce modèle permet de faire est consistant et cohérent, soit plus tard en observant l'utilité de la solution découlant du modèle dans la résolution du problème initial. La seconde forme de validation, c'est-à-dire celle par l'encodage graphique, suppose qu'un mauvais modèle pourrait, tout au moins dans certains cas, mener à une impasse dans la mesure où les données réelles du domaine ne correspondraient pas à ce modèle. Dans ce cas de figure, il est naturellement recommandé d'abandonner le modèle (Sedlmair *et al.*, 2012). L'étude ou le développement des modèles de comportement de code est un domaine de recherche à part entière.

2.4 Modèles de comportement de code

La modélisation du comportement de code fait l'objet de plusieurs travaux (Kwon et Su, 2011; Dupont *et al.*, 2008). Ces travaux, qu'ils soient basés sur l'approche des contraintes sur les données (Ernst *et al.*, 2001; Cicchello et Kremer, 2004), l'approche des machines à états finis (Biermann et Feldman, 1972), ou sur une approche synthétique (Lorenzoli *et al.*, 2008), s'intéressent principalement à la génération de modèles abstraits de comportement de code. Des outils tels que LLVM Machine Code Analyzer peuvent être considérés comme des instanciations de ces modèles. Par ailleurs Dasgupta *et al.* (2019) ont montré que le comportement d'un code s'exécutant sur une architecture cible donnée peut être décrit de manière complète et consistante par trois actions simples comme lire, effectuer une opération élémentaire, écrire. Mais à notre connaissance, aucun modèle de comportement de code proposé jusqu'à présent n'intègre l'architecture cible. Or c'est cette architecture cible, à travers ses registres, qui détermine la sémantique

et plus globalement les paramètres de ces actions élémentaires. Dans le même ordre d'idée, mais à un niveau d'abstraction plus élevé, c'est-à-dire plus proche de l'algorithme dans la taxonomie de Price *et al.* (1993) que de l'architecture physique, il est intéressant d'avoir un modèle permettant de décomposer une fonction complexe en des actions simples similaires à celles évoquées par Dasgupta *et al.* (2019) pour une architecture. Les opérations arithmétiques élémentaires semblent être de bons candidats pour jouer ce rôle vis-à-vis des fonctions vectorielles. De tels modèles sont pratiques du point de vue de la visualisation, ce qui est un critère fondamental. Mais en plus de cet aspect pratique, intégrer l'architecture cible dans la modélisation du comportment d'un code s'éxécutant sur cette architecture nous semble important, puisque cela peut permettre également d'estimer de manière réaliste les paramètres (temps d'exécution, occupation de l'espace sur l'architecture cible, etc.) d'exécution de ce code sur cette architecture.

2.5 Hypothèses et questions de recherches

Comme nous pouvons le constater, la SV (et donc l'InfoViz) est une discipline scientifique assez bien établie et vivante qui est utilisée dans l'aide à la compréhension du code parallèle dont fait partie le code vectoriel. Il est par conséquent naturel de penser à utiliser ses acquis techniques et méthodologiques pour répondre aux besoins de l'aide à la compréhension du code vectoriel. Notre cadre de référence méthodologique en ce qui concerne la visualisation sera donc une synthèse des approches de Munzner (2009) et de celle de Diehl (2007), synthèse reflétée par la Figure 2.13. Cette synthèse ne revient pas à simplement choisir ou pas des éléments de l'approche de Munzner (2009) pour colmater ce qui apparaitrait comme des brèches dans l'approche de Diehl (2007), mais à transposer ces éléments au domaine de la SV ; et cette transposition ne va pas toujours de soi pour ce qui est du comportement de code (Ntang et Lemire, 2021).

2.5.1 Hypothèses

On peut appliquer les techniques de la SV pour aider à la compréhension de la programmation vectorielle, notamment par la conception et l'implémentation d'un outil de visualisation pour aider le programmeur. En particulier, en ce qui a trait au comportement de code, cette application des techniques de la SV suppose la mise au point d'un modèle décrivant ce comportement. L'hypothèse de Dasgupta *et al.* (2019) apparaît alors comme une bonne base pour le développement d'un tel modèle, que ce soit un modèle qui vise le comportement d'un code sur une architecture cible, ou un modèle plus abstrait. Dans les deux cas, d'après la méthodologie de Munzner (2009), si ce modèle est consistant, alors l'encodage visuel qui en découle sera non seulement possible, mais doit permettre effectivement d'aider à la compréhension de la programmation vectorielle. Si l'outil de visualisation obtenu au bout de la démarche respecte ce que Diehl (2007) appelle le pipeline de la visualisation de logiciel, et qui correspond plus largement à la méthodologie présentée par Munzner (2009) ou encore aux quatre étapes fondamentales de toute démarche de visualisation présentée par Spence (2014), alors cet outil permet potentiellement la falsification du programme de recherche que constitue la SV, et donc de l'InfoViz. L'expression programme de recherche est utilisée ici au sens de Lakatos (1976) : c'est-à-dire un ensemble d'outils théoriques et même un ensemble de pratiques dans un domaine de recherche scientifique. Cet outil peut donc sous ces conditions être utilisé pour tester de manière holistique la SV. Nous préférons le terme corroboration à celui de falsification parce que notre intention est de démontrer qu'appliqué au domaine de la programmation SIMD, le programme de recherche correspondant aux outils actuels de la SV marche et non le contraire. Par cette démonstration, nous contribuons à faire du programme de recherche que constitue la pratique de la SV, et par extension de l'InfoViz, un programme non dégénérescent. Bien sûr, un test négatif peut être interprété de

plusieurs façons (par exemple, il peut être dû aux erreurs de conception ou d'implémentation, des erreurs purement de génie logiciel donc), mais un test positif dans ces conditions est une contribution importante au fondement de la SV en tant que paradigme scientifique. Nous pouvons alors à partir de ces hypothèses formuler nos questions opérationnelles.

2.5.2 Questions de recherche

Est-il possible d'utiliser les recettes de la SV, notamment le cadre méthodologique issu de la synthèse des approches de Munzner (2009) et de Diehl (2007) pour concevoir et implémenter un système de visualisation de logiciel pour aider à la compréhension de comportement de codes développés avec des fonctions vectorielles ou *intrinsics* ? Est-il possible de mobiliser les mêmes outils pour concevoir et implémenter un système de visualisation de logiciel pour aider à la compréhension de comportement de ces *intrinsics* ? Au cours du processus de conception et d'implémentation de ce système, est-il possible de développer un modèle pour chacun des domaines manipulés non seulement comme le requiert la méthodologie adoptée (Munzner, 2009; Diehl, 2007), mais un modèle susceptible d'être positionné par rapport aux autres modèles de comportement de code (Kwon et Su, 2011; Dupont *et al.*, 2008) ? Le système de visualisation résultant de ce processus ayant mobilisé les outils de la SV, donc étant une application de la SV au domaine de l'architecture SIMD, peut-on tirer des conclusions quant à la validité de la SV en tant que paradigme scientifique aux conséquences prédictives dans l'aide à la compréhension de comportement de codes et de fonctions SIMD, et donc enrichir les corroborants (falsificateurs potentiels effectivement non falsifiés) de la SV ? Autrement dit, en supposant l'absence d'erreurs relevant du domaine du génie logiciel, observerait-on une différence significative entre les utilisateurs et les non-utilisateurs de cet outil du point de vue de la compréhension de com-

portement des fonctions vectorielles (ou *intrinsics*) ? Pour répondre à la première partie de ces questions, nous développons des modèles respectifs de comportement des fonctions vectorielles et de codes produits à l'aide de ces fonctions, modèles qui sont à la base de l'implémentation de notre prototype appelé SIMDGiraffe.

CHAPITRE III

DÉVELOPPEMENT DES MODÈLES ET IMPLÉMENTATION DES MODULES DE SIMDGIRAFFE

Il s'agit donc, dans la section 3.1, de présenter tour à tour les deux modèles développés. Le premier modèle porte sur le domaine des fonctions vectorielles ou *intrinsics*. Le second modèle porte sur le comportement d'un code s'exécutant sur une architecture cible donnée. Ces modèles sont du point de vue de la visualisation d'importantes contributions comme le souligne Munzner (2008). La possibilité de concevoir et d'implémenter des modules du prototype SIMDGiraffe chacun basé sur un de ces modèles en est une validation respective suivant la méthodologie de Munzner (2009). Mais ces modèles ont une portée beaucoup plus générale, qui va au-delà de la visualisation. Nous donnons pour le modèle du domaine des fonctions vectorielles en particulier un aperçu de comment il pourrait être transformé en une ontologie d'application et ainsi être utilisé pour effectuer des raisonnements sur ces fonctions vectorielles par exemple. Dans la section 3.2, nous présentons comment en partant de chacun de ces modèles sont conçus et implémenté chacun des modules correspondants de SIMDGiraffe. Le module 2 vise à aider à la compréhension du fonctionnement des *intrinsics* alors que le module 1 est un support dans la compréhension du comportement d'un code créé avec les *intrinsics*.

3.1 Développement de modèles : modèle des fonctions vectorielles et modèle de comportement d'un code exécuté sur une architecture cible donnée

Ces modèles qui cristallisent à la fois le problème à résoudre à travers la visualisation et le domaine de définition de ce problème doivent être mis au point dès le début du processus de visualisation. Bien entendu, comme toutes les étapes de ce processus de visualisation, le développement d'un modèle est itératif et incrémental, avec des allers et retours entre les étapes en aval et les étapes en amont. En raison des ces allers et retours entre le l'étape du modèle et les autres étapes en aval, la capacité de concevoir et d'implémenter un prototype d'outil de visualisation basé sur un modèle donné est une validation du modèle en question (Sedlmair *et al.*, 2012; Munzner, 2009). Pour le modèle des fonctions vectorielles en particulier, le fait de valider expérimentalement le module de SIMDGiraffe qui repose dessus renforce encore plus la validité de ce modèle.

3.1.1 Développement d'un modèle des fonctions vectorielles et applications [1]

Ce modèle permet d'exprimer les fonctions vectorielles en partant des opérateurs scalaires arithmétiques élémentaires, et en utilisant le formalisme des espaces vectoriels réels. Un des intérêts du modèle est qu'il nous permet alors d'utiliser les propriétés de ces structures algébriques connues pour décrire les manipulations à effectuer sur les données afin de leur donner du sens, afin d'expliquer les relations qui existent entre ces données. Ainsi donc, soit E un ensemble quelconque, n un entier naturel non nul et $F = (E, \mathbb{R}^n)$ l'ensemble des fonctions de E dans $\mathbb{R}^n$. F est un espace vectoriel (Chambadal et Ovaert, 1966). Si $n = 1$, F est l'espace vectoriel des fonctions qui à un élément de E associent un réel, c'est-à-dire l'espace

1. Les idées développées dans cette partie font l'objet d'un article à venir qui est intitulé : « SIMDGiraffe : Capturing and Visualizing the Expert's Mind to Understand SIMD Functions ».

vectoriel des fonctions réelles (ou encore scalaires) définies sur E. Puisque ce sont les fonctions vectorielles qui nous intéressent, nous considérons donc que $n \geq 2$. Étant donné un élément f de F et un élément x de E, on a $f(x) = y \in \mathbb{R}^n$ et $y \in \mathbb{R}^n \Rightarrow y = (y_0, ..., y_{n-1})$. Rappelons que $\mathbb{R}^n$ et $\mathbb{R}$ sont également des espaces vectoriels. Par conséquent, $(y_0, ..., y_{n-1})$ sont les composantes du vecteur $y = f(x)$ dans $\mathbb{R}^n$. Mais dans l'espace vectoriel $\mathbb{R}$, y_i est lui-même un vecteur. Supposons que la famille de vecteurs $(y_i)_{i\in[0,n-1]}$ soit une famille de vecteurs indépendants, c'est-à-dire que $\sum_i a_i y_i = 0 \Rightarrow \forall i, a_i = 0$; On obtient alors que $(y_i)_{i\in[0,n-1]}$ est une base de $\mathbb{R}^n$. Dans ce cas, soit p_i le projecteur de $\mathbb{R}^n$ de direction la droite ${y_i}^\perp$ si $n = 2$, le plan ${y_i}^\perp$ si $n = 3$ ou l'hyperplan ${y_i}^\perp$ si $n > 3$. Par définition, p_i préserve, pour tout vecteur de $\mathbb{R}^n$, la composante i, et pour $j \neq i$, la composante j vaut le vecteur nul. Autrement dit on a :

$$p_i(y) = y_i; \tag{3.1}$$

et puisque $y = f(x)$, on obtient :

$$p_i f(x) = y_i. \tag{3.2}$$

L'équation (3.1) signifie que $p_i(\mathbb{R}^n) = \mathbb{R}$. Nous savons aussi que

$$\mathbb{R}^n = \sum_{i=0}^{n-1} \bigoplus p_i(\mathbb{R}^n). \tag{3.3}$$

Dans (3.2), p_i ne dépend pas de x, on peut donc écrire $p_i f = f_i$. L'équation (3.2) devient $f_i(x) = y_i$, mais $f(x) = (y_0, ..., y_{n-1}) = (f_0(x), ..., f_{n-1}(x))$ et à cause de (3.3)

$$f(x) = \sum_{i=0}^{n-1} f_i(x) \qquad \forall x \in E. \tag{3.4}$$

Par définition, (3.4) signifie $f = \sum_i f_i$. Il convient de noter que $(f_i)_{i\in[0,n-1]}$ forment une famille de vecteurs indépendants, car si on a $\sum_{i=0}^{n-1} f_i = 0$, alors $\forall x, \sum_{i=0}^{n-1} a_i f_i(x) = 0$, soit $\sum_{i=0}^{n-1} a_i y_i = 0$ et comme la famille $(y_i)_{i\in[0,n-1]}$ est libre, on a forcément $\forall i \in [0, n-1]$, $a_i = 0$.

Nous venons de montrer que sous notre hypothèse, on peut décomposer f en n fonctions indépendantes f_i avec $f_i(x) = y_i$. Nous pouvons faire plusieurs remarques. i) Si nous avons une architecture avec des registres de taille n et une instruction avec p paramètres formels en entrée, $E = (\mathbb{R}^n)^p = \mathbb{R}^{np}$. ii) Cette hypothèse est fondamentale et nécessaire pour tout ce qui va suivre, jusqu'à la représentation graphique. Par exemple, si $n = 2$ et $p = 2$, soit $x_1, x_2, y \in \mathbb{R}^2$ tel que $x_1 = (x_1^1, x_1^2), x_2 = (x_2^1, x_2^2)$ et $y = (y_1, y_2)$. Soit $x = (x_1, x_2)\,;\, x \in \mathbb{R}^4$. Soit f une fonction vectorielle hypothétique définie de $\mathbb{R}^4$ dans $\mathbb{R}^2$ par $f(x) = y$ telle que

$$y_1 = x_1^1 + x_2^1 \; et \; y_2 = \begin{cases} x_1^2 - x_2^2 \; si \; y_1 \in \mathbb{N} \; et \; y_1 \; est \; pair \\ \\ x_1^2 + x_2^2 \; sinon \end{cases} . \tag{3.5}$$

Alors, dans l'équation (3.5), on ne peut pas avoir une projection de $f(x)$ sur une de ses dimensions avec p_i, car y_1, y_2 ne sont pas indépendants, donc $p_i f(x)$ dépend de x. Nous ne pouvons donc pas décomposer la fonction f telle que définie en la somme de deux fonctions indépendantes puisque notre hypothèse n'est pas vérifiée. Par conséquent, nous ne serions pas en mesure de représenter cette fonction hypothétique dans SIMDGiraffe. Cependant, à notre connaissance, il n'existe pas de telles fonctions vectorielles, en particulier dans l'ensemble de données utilisé, c'est-à-dire les *intrinsics* de l'architecture Intel® AVX-512. Mais même si de telles fonctions devaient exister dans un jeu de données, notre hypothèse et donc les résultats qui en découlent restent valables pour les autres instructions ; ces autres instructions forment sinon la totalité, du moins l'écrasante majorité des instructions possibles. Ce modèle permet de conceptualiser le domaine à l'étude, qui est celui des fonctions vectorielles ou *intrinsics*. Le formalisme que ce modèle apporte permet de formuler de manière plus simple et précise le problème à résoudre au regard du domaine à l'étude. Ce formalisme permet également d'abstraire les données et les opérations nécessaires sur ces données pour résoudre le problème. Il convient de noter que l'intérêt de ce modèle va au-delà de la visualisation ; il peut par exemple être transformé en une ontologie d'application.

TABLEAU 3.1 – Exemple de fonctions vectorielles (nous ajoutons les attributs rootFunction, prefixFunction et suffixFunction pour décrire respectivement la racine du nom de la fonction, son préfixe et son suffixe).

ID	nameFonction	rootFunction	prefixFunction	suffixFunction	instruction	nbrOperand	returnType
F1	_mm_shuffle_epi32	shuffle	_mm	_epi32	pshufd	2	__m128i
F2	_mm512_mask_add_ps	_mask_add	_mm512	_ps	vaddps	4	__m512

TABLEAU 3.2 – Exemple d'opérandes (nous laissons les valeurs vides, mais dans l'utilisation ces valeurs seraient indiquées).

ID	operatorName	operator	nbrOfParameters
T1	addition	+	2
T2	permutation	Mov2	2
T3	inversion	Inv	1
T4	indexation	Idx	2

3.1.1.1 Indication sur la transformation du modèle en une ontologie d'application

L'intérêt d'une telle transformation serait d'utiliser les outils *Resource Description Framework Schema* (RDFS) et *Web Ontology Language* (OWL) pour effectuer des raisonnements sémantiques sur les fonctions vectorielles et leur comportement. Cela peut se faire sous un environnement comme PROTEGE, ou même dans le cadre d'une application utilisateur développée dans un environnement comme MOBI. Pour construire l'ontologie, on peut procéder par abstraction croissante et successive (plusieurs itérations) en partant de la couche *Resource Description Framework* (RDF) (donnée) à la couche OWL en passant par la couche RDFS. C'est une approche *bottom-up*, plus rigoureuse, qui permet d'éviter certaines erreurs de transposition du modèle. L'ontologie que nous suggérons de construire

TABLEAU 3.3 – Exemple d'opérandes (nous laissons les valeurs d'opérandes vides, mais dans l'utilisation ces valeurs seraient indiquées).

ID	nameOperand	sizeOperand	valueOperand
D1	__m128i	128	
D2	__m512	512	
D3	__mmask16	0	

TABLEAU 3.4 – Exemple de champs d'opérandes vectoriels (nous laissons les valeurs de champs vides, mais dans l'utilisation ces valeurs seraient indiquées).

ID	typeOfField	valueOfField
C1	int	
C2	string	
C3	byte	
C4	bit	

TABLEAU 3.5 – Traduction des cellules de la première colonne en triplets RDF.

subject	int : func1	int : func2	int : operan1	int : operan2	int : operan3	int : operaF1
predicate	int : funcId	int : funcId	int : operanId	int : operanId	int : operanId	int : operaFId
object	F1	F2	D1	D2	D3	C1
subject	int : operaF2	int : operaF3	int : operaF4	int : operat1	int : operat2	int : operat3
predicate	int : operaFId	int : operaFId	int : operaFId	int : operatId	int : operatId	int : operatId
object	C2	C3	C4	T1	T2	T3

est une ontologie d'application (les deux autres types d'ontologies étant les ontologies globales et les ontologies de domaine). Cette ontologie est extensible sans perdre en clarté et en cohérence, toutes qualités que doivent vérifier une ontologie. Compte du domaine et du modèle développé, les termes importants doivent

TABLEAU 3.6 – Description des attributs des fonctions exemples avec RDF.

subject	predicate	object
int : func1	int : nameFunction	_mm_shuffle_epi32
int : func1	int : rootFunction	shuffle
int :func1	int : prefixFunction	_mm
int :func1	int : suffixFunction	_epi32
int :func1	int : instruction	pshufd
int :func1	int : nbrOperand	2
int : func1	int : returnType	__m128i
int : func2	int : nameFunction	_mm512_mask_add_ps
int : func2	int : rootFunction	_mask_add
int :func2	int : prefixFunction	_mm512
int :func2	int : suffixFunction	_ps
int :func2	int : instruction	vaddps
int :func2	int : nbrOperand	4
int :func2	int : returnType	__m512

TABLEAU 3.7 – Description des attributs des opérandes exemples avec RDF.

subject	int : operan1	int : operan1	int : operan1	int : operan1	int : operan1	int : operan1
predicate	int : nameOperand	int : sizeOperand	int : valueOperand	int : nameOperand	int : sizeOperand	int : valueOperand
object	__m128i	128		__m512	512	

tenir compte des concepts de fonctions vectorielles (ou *intrinsics*), d'opérandes, de champs d'un opérande et d'opérateurs. Dans notre modèle, le résultat est juste une sorte d'opérande. À partir de ces concepts, nous pouvons figurer la première représentation des triplets RDF représentant la couche donnée de notre modèle dans le formalisme des ontologies. Pour bien comprendre le modèle, nous partons de représentations tabulaires des Tableaux 3.1, 3.2, 3.3 et 3.4. Il faut comprendre

TABLEAU 3.8 – Spécification de la nature des objets en RDF (à titre d'illustration, les trois dernières cellules signifient que func2 est un type RDF function. Bien sûr pour l'instant cela n'a aucune sémantique).

subject	int : operaF1	int : operaF2	int : operaF3	int : operaF4	int : operat1	int : operat2
predicate	RDF : type	RDF : type	RDF : type	RDF : type	RDF : type	RDF : type
object	int : operandField	int : operandField	int : operandField	int : operandField	int : operator	int : operator
subject	int : operat3	int : operan1	int : operan2	int : operan3	int :func1	int : func2
predicate	RDF : type	RDF : type	RDF : type	RDF : type	RDF : type	RDF : type
object	int : operator	int : operand	int : operand	int : operand	int : function	int : fucntion

TABLEAU 3.9 – Exemple de spécification du caractère propriété (par opposition à classe) des attributs en RDF.

subject	predicate	object
int : nameFunction	RDF : type	RDF : Property
int : rootFunction	RDF :type	RDF :Property
int : prefixFunction	RDF :type	RDF :Property
int : suffixFunction	RDF : type	RDF : Property
int : instruction	RDF : type	RDF : Property
int : nbrOperand	RDF : type	RDF : Property
int : nameOperand	RDF : type	RDF : Property
int : sizeOperand	RDF : type	RDF : Property
int : valueOperand	RDF : type	RDF : Property

ici chaque ligne comme un enregistrement portant sur un individu et les colonnes comme des propriétés. L'utilisation d'une nouvelle fonction ou d'un nouvel opérateur, ou même simplement l'utilisation d'une nouvelle valeur d'opérande revient à insérer un nouvel enregistrement dans le tableau correspondant. Ainsi, ce qui est susceptible d'évoluer (et seulement quantitativement, en nombre) ce sont les

TABLEAU 3.10 – Spécification des deux relations en RDF.

Subject	int : isPartOf	int : haveFirstFactor
Predicate	RDF : type	RDF : type
Object	RDF : Property	RDF : Property

TABLEAU 3.11 – Traduction des deux relations à l'aide de RDF.

Subject	int : operaF1	int : operat1
Predicate	int : isPartOf	int : haveFirstFactor
Object	int : operand1	int : operaF1

TABLEAU 3.12 – Spécification des objets du modèle en RDFS ; function, operand ont une sémantique : ce sont des classes RDFS.

Subject	int : operand	int : operandField	int : operator	int : function
Predicate	RDF : type	RDF : type	RDF : type	RDF : type
Object	RDFS : Class	RDFS : Class	RDFS : Class	RDFS : Class

TABLEAU 3.13 – Spécification de la nature des relations (ensembles de départ et ensembles d'arrivée).

Subject	int : isPartOf	int : isPartOf	int : haveFirstFactor	int : haveFirstFactor
Predicate	RDFS : domain	RDFS : range	RDFS : domain	RDFS : range
Object	int : operatorField	int : operand	int : operator	int : operatorField

fonctions, les opérandes, etc. Cette évolution se traduit toujours par une insertion d'une nouvelle ligne dans le tableau correspondant et revient en fait au peuplement de l'ontologie. Cela n'affecte pas le modèle transposé, bien au contraire c'est le but même du modèle : être peuplé pour être utilisé dans une application ou dans un but de raisonnement. Rappelons qu'à ce niveau, il s'agit de représenter (trans-

poser) les concepts du domaine et de permettre de décrire les instances de données et les interactions entre ces données. Cette description des instances doit se faire en reliant ces concepts par des relations (niveau RDF). Nous allons retranscrire les tables qui sont en fait l'expression de relations (au sens du modèle relationnel) en RDF pour représenter les données (les deux *intrinsics* `_mm_shuffle_epi32` et `_mm512_mask_add_ps`, les trois opérandes, les quatre champs d'opérandes, etc.). En désignant l'espace de nom par *int* (pour Intel®), on obtient donc à la première itération les Tableaux 3.5, 3.6, 3.7, 3.8 et 3.9.

Après cette traduction non exhaustive en RDF des attributs pour les données choisies, nous pouvons maintenant énoncer les relations et les traduire sous forme de propriétés. Nous pouvons déjà voir quelques relations comme le champ opérande de type *int* (C1) est une partie de l'opérande `__m512` (D1) ; l'opérateur addition (T1) a pour premier membre le champ dont l'identifiant est C1, etc. On peut traduire sémantiquement ces deux exemples de relation en RDFS (avec RDF) comme présenté dans les Tableaux 3.10 et 3.11. Nous allons donc monter en niveau d'abstraction en définissant ces relations à l'aide de la sémantique RDFS à travers les concepts de domain et range comme le montre les Tableaux 3.12 et 3.13. Une fois ces schémas mis en place, on peut encore monter au dernier niveau d'abstraction, en décrivant (construisant) de nouvelles classes à l'aide des classes existantes avec le formalisme OWL. Nous pouvons ainsi définir par exemple les fonctions prenant deux opérandes :

```
int : nbrOperand a OWL:DatatypeProperty;
            RDFS:range xsd:integer.
int : twoOperandFunction OWL:equivalentClass
            [a OWL: Restriction;
                        OWL:onProperty int:nbrOperand;
                        OWL:hasValue 2]
```

Ainsi, notre modèle pourrait être très aisément traduit en ontologie et être intégré dans une application intelligente, ou même permettre de faire des inférences sur les fonctions vectorielles.

3.1.2 développement d'un modèle de comportement d'un code exécuté sur une architecture cible donnée [2]

Le comportement d'un code, qu'il soit vectoriel ou non, est dépendant de l'environnement physique dans lequel ce code est effectivement exécuté. Cet environnement physique est principalement matérialisé par l'architecture de l'ordinateur. C'est d'ailleurs un type d'architecture bien précis, à savoir l'architecture vectorielle, qui rend possible l'exécution de code vectoriel. Nous insistons sur cet aspect architectural, car après tout, c'est la condition première de la vectorisation. En effet, sans une architecture matérielle appropriée, en particulier la présence de registres vectoriels, la vectorisation n'est pas possible. Pour le comportement d'un code à l'exécution, on peut utiliser l'instrumentation. Mais l'instrumentation est généralement une source de biais puisqu'elle peut modifier le comportement d'un code à l'exécution (Diehl, 2007). De plus, l'instrumentation rend peu claire la séparation entre référents et attributs. Cependant, nous savons (2.3.3, page 46) que cette séparation est la norme pour d'autres domaines de la visualisation d'information (Purchase *et al.*, 2008). La modélisation du comportement de code a fait l'objet de plusieurs travaux (Kwon et Su, 2011; Dupont *et al.*, 2008). Ces travaux, qu'ils soient basés sur l'approche des contraintes sur les données (Ernst *et al.*, 2001; Cicchello et Kremer, 2004), l'approche des machines à états finis (Biermann et Feldman, 1972), ou sur une approche synthétique (Lorenzoli *et al.*, 2008), s'inté-

---

2. Les idées développées dans cette partie font l'objet d'un article publié intitulé « SIMDGiraffe : Visualizing SIMD Functions », et présenté à IVAPP 2021 (DOI :10.5220/0010195201470154).

ressent principalement à la génération de modèles de comportement de code. Des outils tels que LLVM Machine Code Analyzer peuvent être considérés comme des instanciations de ces modèles. Nous avons besoin d'un modèle opérationnel pour générer les attributs décrivant le comportement d'un code s'exécutant sur une architecture cible donnée. Cela nous rapprochera de la norme des autres domaines de la visualisation d'information. Ce modèle est basé sur l'hypothèse et l'observation que tout programme est déterminé, du point de vue de son comportement, par une instance de ce programme sur une architecture physique et uniquement. Nous utilisons le terme architecture pour désigner la machine physique et ses primitives de base qui permettent la manipulation du matériel. Sous forme fonctionnelle, on peut écrire que le comportement d'un programme à l'exécution (PE) est

$$PE = F(S, H) \tag{3.6}$$

où S est le code source ou le logiciel, et H est l'architecture ou le matériel sur lequel ce code source s'exécute. La fonction F peut être décomposée en fonctions élémentaires, chacune correspondant à une occurrence d'une instruction dans un registre utilisé par le programme pendant son exécution. Tout le reste de la mémoire qui ne fait pas partie des registres est vu comme un seul registre particulier. Ce formalisme permet, au moins au niveau du langage machine, de décrire la sémantique d'un code de manière cohérente (Dasgupta *et al.*, 2019). L'équation 3.6 est alors une agrégation de ces fonctions élémentaires. En faisant l'hypothèse que l'agrégation des fonctions élémentaires restitue le comportement global du code, la sortie de cette équation correspond aux attributs utilisés pour caractériser le comportement d'un code S sur l'architecture H. Ce comportement est décrit en termes d'occupation de la mémoire, d'opérations d'entrée/sortie effectuées par le code, de séquences de modifications effectuées par le code sur les registres, c'est-à-dire les séquences de calcul et de contrôle, ou encore de performances d'exécution du code en termes de durée, etc. Dans cette équation, F est défini sur $S \times H$ où S

est le code source vectoriel et H est l'architecture vectorielle. Nous pouvons tirer parti du fait que pour un *intrinsic* ou son équivalent en assembleur, ses instances dans le code source S sont en nombre fini, tout comme les registres de H. Par exemple, il y a seize registres SIMD de 512 bits en mode 64 bits sur la génération AVX-512 (Intel, 2011) et mieux encore, tous les registres ne sont pas utilisés lors de l'exécution d'un programme donné. Nous décomposons ensuite S en une série d'instances de chacune de ses instructions. Nous exploitons le fait que les fonctions vectorielles ont un équivalent en assembleur, et donc ce sont ces équivalents qui apparaissent dans cette décomposition de S. De cette manière, nous assimilons S à ces instances. L'architecture H est également assimilée aux registres utilisés par le code S lors de son exécution. Nous définissons la relation R de S sur H par sRh si et seulement si s utilise h pendant l'exécution du code source S sur l'architecture H. Le graphe de R est un sous-ensemble du tableau matriciel $S \times H$. La restriction de F à R se décompose alors en fonctions élémentaires f_r^i, et la sortie de chacune de ces fonctions décrit le comportement de l'instance i d'une instruction vectorielle sur un registre r. Le tableau matriciel est peuplé en décrivant chaque élément (i, r) de R avec la sortie de f_r^i. Dans la cellule correspondant à cet élément, on place la description du comportement de l'instruction sur le registre. Finalement, on obtient un tableau à double entrée dont les lignes sont des instances, les colonnes des registres, et la cellule (i, r) est occupée par la sortie de f_r^i correspondant à l'instance i et au registre r si ce couple est dans le graphe de R ($(i, r) \in R$) ou rien si ce couple n'est pas dans le graphe de R. Chaque colonne est ordonnée puisque les instances apparaissent dans l'ordre dans lequel elles utilisent le registre correspondant. Cet ordre est total, car deux instances ne peuvent pas utiliser le même registre en même temps. Cet ordre est obtenu à partir de la description en sortie de la fonction F. On a donc une matrice dont la cellule (i, r) est un objet qui décrit l'interaction de l'instance i sur le registre r si ce couple est dans le graphe de R et 0 ou *null* si ce couple n'est pas dans le

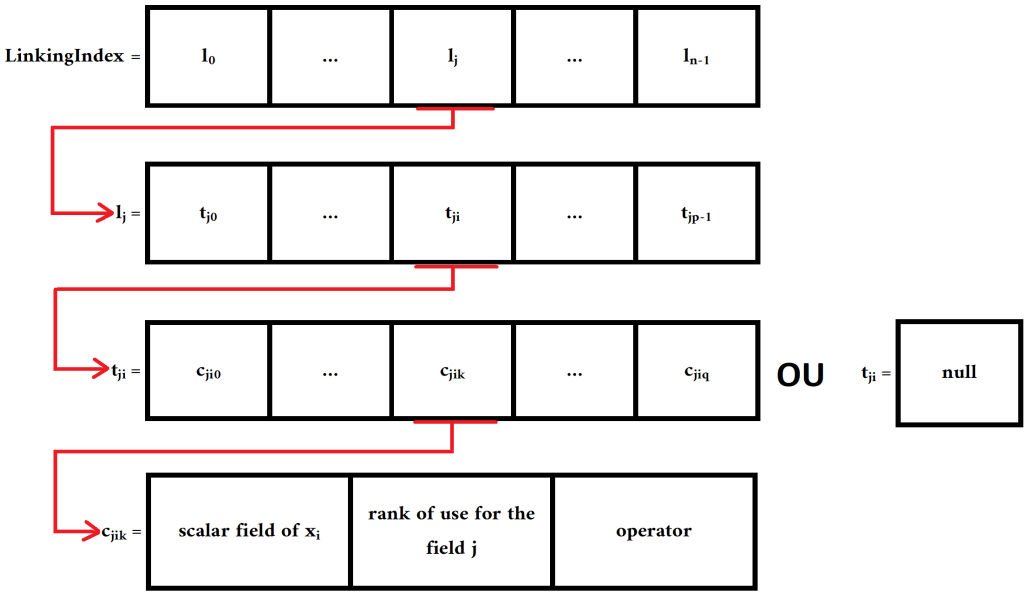


FIGURE 3.1 – Visualisation de *LinkingIndex*.

graphe de R.

3.2 SIMDGiraffe : conception et réalisation d'un outil de visualisation pour aider à comprendre les fonctionnements respectifs d'un *intrinsic* (module 2) et d'un code s'exécutant sur une architecture cible (module 1)

Une fois les deux modèles acquis, nous pouvons effectivement passer à la conception et à l'implémentation des deux modules correspondants proprement dits. Il convient de noter tout de même que la séparation entre le développement d'un modèle et la conception du module correspondant est une séparation purement logique et non dans le temps. L'ordre de présentation des deux modules est aussi plutôt logique que chronologique. Ainsi, nous présentons d'abord le module permettant de comprendre le fonctionnement des *intrinsics* avant d'aborder celui dédié à l'aide à la compréhension de code créé en utilisant les *intrinsics*.

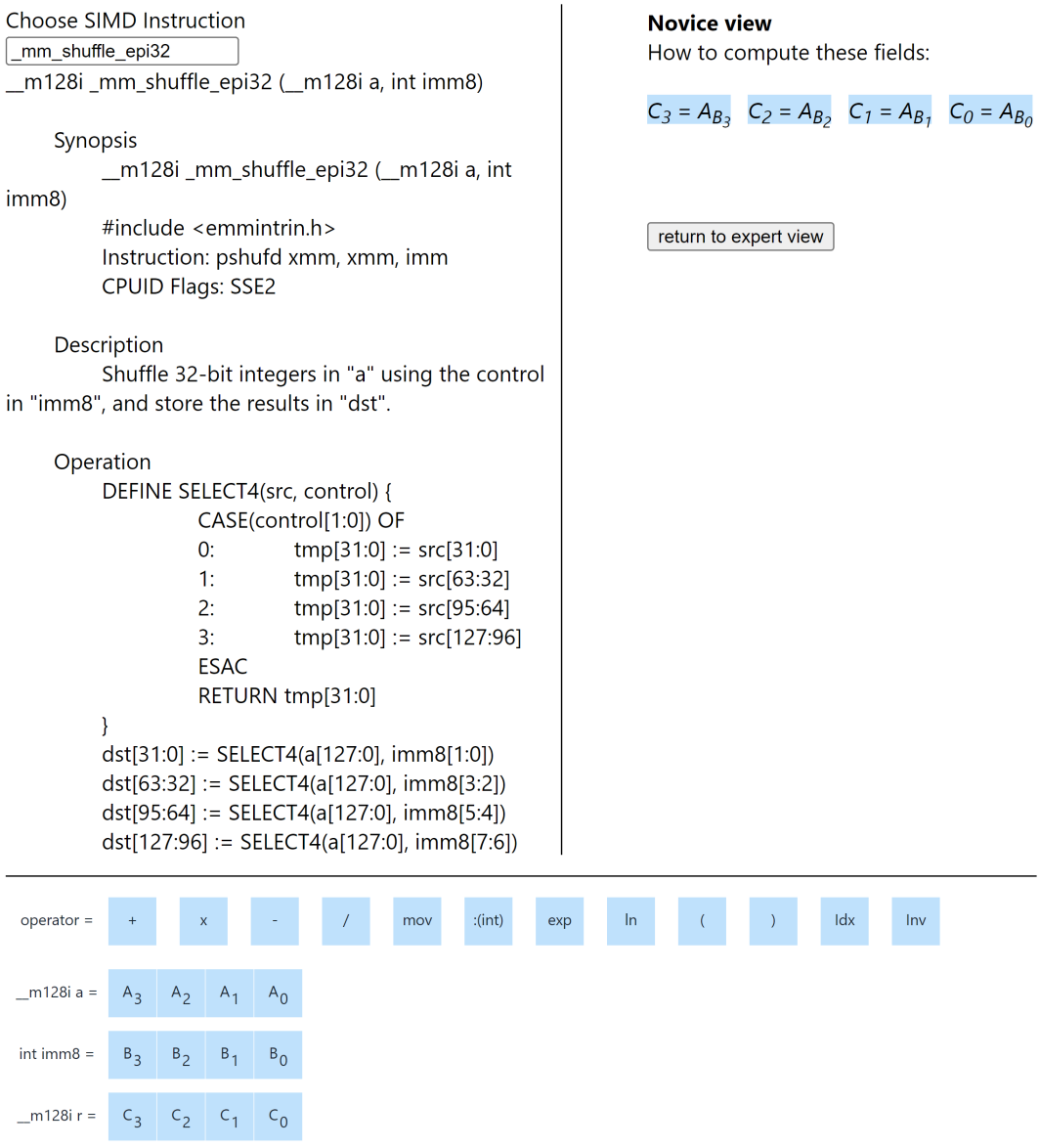


FIGURE 3.2 – Vue Novice `_mm_shuffle_epi32` du module 2 de SIMDGiraffe.

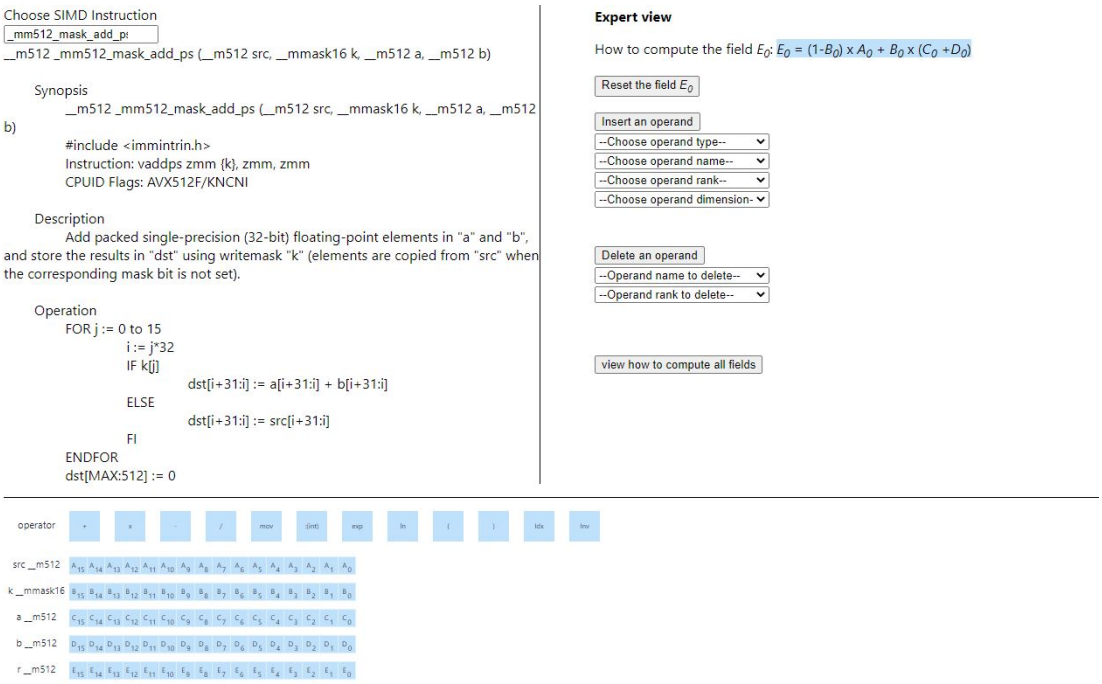


FIGURE 3.3 – Vue Expert `_mm512_mask_add_ps` du module 2 de SIMDGiraffe.

3.2.1 Conception et mise en œuvre d'un outil de visualisation afin de supporter la compréhension de comportement des fonctions vectorielles ou *intrinsics*[3]

Il est important de préciser ce que signifie comprendre le comportement d'un *intrinsic*, le problème que pose cette compréhension, avant de proposer une approche pour résoudre ce problème.

3.2.1.1 Spécification du problème et identification des données à utiliser

Grâce à notre modèle, le problème à résoudre et qui consiste à aider un novice à comprendre le comportement d'une fonction vectorielle ou *intrinsic*, peut se

3. Les idées développées dans cette partie font l'objet d'un article à venir qui est intitulé : « SIMDGiraffe : Capturing and Visualizing the Expert's Mind to Understand SIMD Functions ».

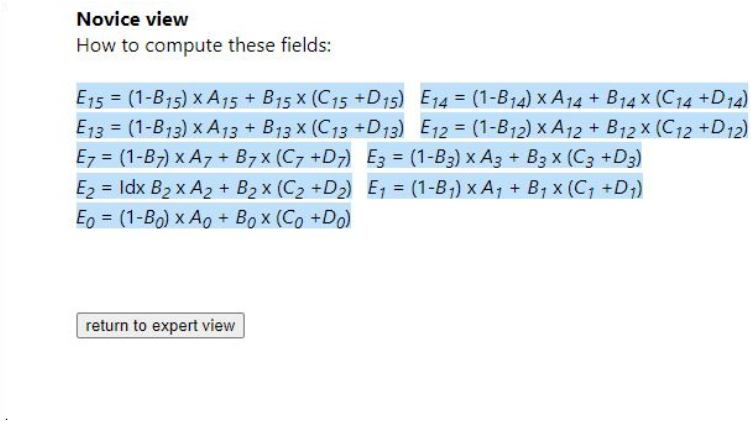


FIGURE 3.4 – Vue Novice `_mm512_mask_add_ps`.

formuler de manière plus simple et précise. Comprendre le comportement d'un *intrinsic* revient à comprendre le lien entre ses opérandes en entrée et ses résultats en sortie. Par exemple, considérons l'expression définie par

$$z = \_mm512\_add\_epi8(x, y). \tag{3.7}$$

Quel est le lien logique et arithmétique entre les entités x, y et l'entité z dans (3.7) ? Les éléments de réponses disponibles sur le site interactif d'Intel® (Intel, 2018) ne sont pas toujours faciles à comprendre, surtout pour une personne qui n'est pas déjà familiarisée avec la manipulation des *intrinsics*. Il peut donc être nécessaire d'avoir recours à un expert déjà habitué à manipuler ces *intrinsics* pour aider le novice à comprendre le lien entre les entrées et la sortie correspondante. À ce niveau, un nouveau problème se pose. Si l'explication de l'expert consiste à ajouter du texte, écrit ou verbal, au texte que constitue l'explication d'Intel® et le pseudocode, cela n'aidera pas forcément le novice à progresser. En pratique, l'expert appuiera son texte par un schéma pour faciliter la compréhension. Mais nous savons (2.1.2.1, page 26) que ce schéma peut être entaché d'erreurs, manquer de clarté et être biaisé pour le novice (Nathan *et al.*, 2001). Il est donc intéressant de proposer un cadre qui permette à l'expert de s'exprimer, tout en le contraignant

à une explication accessible pour le novice. D'une part, ce cadre doit à la fois structurer l'explication de l'expert et en même temps faciliter cette explication. D'autre part, il est nécessaire de s'assurer de l'utilité et de l'utilisabilité de ce cadre pour aider le novice à comprendre les *intrinsics*. Pour y parvenir, une étape importante est l'élucidation des considérations liées aux données à utiliser en entrée, c'est-à-dire leur identification. En général, pour acquérir les données décrivant le comportement d'un code, on peut recourir à l'instrumentation ou à l'exécution abstraite de ce code (Diehl, 2007). Évidemment, ceci n'est pas possible pour un *intrinsic*, car il est l'équivalent d'une instruction assembleur. Dans le cas des *intrinsics*, en particulier ceux de l'Intel® AVX-512, les données qui les décrivent et les caractérisent sont clairement identifiées. La première donnée est celle que l'on veut expliquer, c'est-à-dire les *intrinsics* eux-mêmes. Nous utilisons donc le jeu de données des instructions Intel® disponible en ligne à l'adresse `https://software.intel.com/sites/landingpage/IntrinsicsGuide/files/data-3.4.6.xml` pour générer les données brutes. Dans ce jeu de données d'instructions, on trouve les *intrinsics* eux-mêmes, leur structure, le pseudo-code explicatif ; le tout au format XML. Le pseudo-code, bien que présentant une régularité, reste comparable à un langage naturel ; il ne s'agit pas d'un code exécutable. De fait, l'extraction automatique des connaissances contenues dans ce pseudo-code est difficile (Lemire *et al.*, 2019). Si cette extraction était facile, une solution entièrement automatique, c'est-à-dire sans l'intermédiaire d'un expert, serait plus adéquate (Sedlmair *et al.*, 2012). Mais il s'avère que la logique permettant d'interpréter ce pseudo-code et le texte qui l'accompagne est dans la tête de l'expert. Un outil visuel est donc indiqué (Sedlmair *et al.*, 2012) pour donner un cadre d'expression à la fois contraignant et convivial à cette logique. Nous ajoutons à cette première donnée des opérateurs arithmétiques élémentaires sous la forme d'un fichier dans lequel figurent les opérateurs arithmétiques scalaires pris en charge. Les métadonnées sont constituées de la logique explicative possédée par l'expert et des règles de

l'arithmétique et de l'algèbre élémentaires. Nous générons les données d'entrée pour SIMDGiraffe, c'est-à-dire les données brutes du point de vue de la visualisation de l'information, à partir de ces données et métadonnées. Dans la suite de ce document, lorsque nous faisons référence aux données sans autre précision, nous nous référons à ces données générées. Une fois les données identifiées, il faut alors trouver un type abstrait de données (TAD) approprié pour représenter de manière consistante le modèle.

3.2.1.2 Type abstrait de données et représentation du modèle

Le type abstrait de données doit à la fois représenter de manière consistante le modèle et capturer la logique arithmétique et algébrique qui relie les opérandes et le résultat. Il doit également être facilement traduisible en une représentation visuelle et graphique. Cette dernière spécification rend par exemple l'utilisation d'une ontologie applicative peu adéquate, bien qu'elle puisse être intéressante comme représentation pour faire du raisonnement sémantique sur les *intrinsics*. Nous optons donc pour un tableau pour coder la logique d'explication. Un tableau, qui préfigure un vecteur, semble le mieux adapté aux représentations graphiques à structure géométrique linéaire. Appelons ce tableau $LinkingIndex$.

Supposons une architecture SIMD dont la taille de registre est n et une fonction vectorielle f qui prend p paramètres. Appelons r le vecteur renvoyé par cette fonction f. Cela signifie que

$$x = (x_0, ..., x_i, ..., x_{p-1}) \quad \text{dans (3.4), et}$$
$$r = (r_0,\ ...,\ r_j,\ ...,\ r_{n-1}).$$

Rappelons que dans ce cas $E = (\mathbb{R}^n)^p = \mathbb{R}^{np}$. De (3.4), on a :

$$\begin{aligned} f(x) &= (f_0(x),\ ...,\ f_j(x),\ ...,\ f_{n-1}(x)) \\ &= (f_0(x_0,\ ...,\ x_i,\ ...,\ x_{p-1}),\ ...,\ f_j(x_0,\ ...,x_i,...,x_{p-1}),...,f_{n-1}(x_0,...,x_i,...,x_{p-1})) \\ &= (r_0,...,r_j,...,r_{n-1}) \\ &= r. \end{aligned}$$

Il apparaît alors que $LinkingIndex$ (Figure 3.1) est de taille n, égale au nombre de champs dans le vecteur résultat et est défini par : $LinkingIndex = [l_0, ..., l_j, ..., l_{n-1}]$. Un élément l_j de $LinkingIndex$ est un tableau avec p éléments où p est le nombre d'opérandes de f ; c'est-à-dire que p est le nombre de paramètres de f. $l_j = [t_{j_0}, ..., t_{j_i}, ..., t_{j_{p-1}}]$. Chaque élément t_{j_i} de l_j est aussi un tableau. t_{j_i} décrit la contribution de l'opérande x_i au champ j du vecteur résultat r. Si l'opérande x_i ne contribue pas au calcul du champ j du vecteur résultat r, t_{j_i} est un tableau vide. Un champ donné de l'opérande x_i peut contribuer avec plusieurs instances au calcul du résultat d'un champ donné du vecteur résultat r, c'est-à-dire qu'il peut être utilisé plusieurs fois à des positions différentes. Si l'opérande x_i contribue au champ j du vecteur résultat $q+1$ fois, c'est-à-dire avec $q+1$ instances, alors t_{j_i} est un tableau de longueur $q+1$. On a alors $t_{j_i} = [c_{j_{i_0}}, ..., c_{j_{i_k}}, ..., c_{j_{i_q}}]$. $c_{j_{i_k}}$ est un tableau à trois éléments. $c_{j_{i_k}}[0]$ indique le champ scalaire du vecteur d'opérande x_i qui est utilisé. $c_{j_{i_k}}[1]$ indique à quel rang, à quelle position ce champ scalaire est utilisé dans le calcul du champ j du vecteur résultat r. $c_{j_{i_k}}[2]$ indique l'opérateur scalaire qui précède ce champ scalaire de l'opérande x_i dans la formule de calcul du champ j du vecteur résultat r. Il convient de noter que la construction de cette structure de données, $LinkingIndex$, pose un défi technique important. En effet, il est difficile de déterminer le nombre de champs d'un vecteur donné. Le nombre de bits d'un opérande ou du vecteur résultat peut être obtenu en décodant ce vecteur. Par exemple, un vecteur comme $\_\_m512$ a une longueur de 512 bits. Mais à l'usage, les champs de ce vecteur peuvent être une entité codée sur 1, 8, 16, 32 ou

64 bits et donner ainsi un nombre de champs de 512, 64, 32, 16 ou 8. Avec cette représentation on peut passer à la phase d'encodage visuel et à la conception de l'aspect interactif de l'outil.

3.2.1.3 Conception de l'encodage visuel et des interactions

En termes d'encodage visuel, il est souhaitable d'utiliser la connectivité, la proximité, la similarité des couleurs, etc., par ordre décroissant de préférence, pour représenter les liens entre les éléments (Diehl, 2007; Wertheimer, 2012). Ainsi, selon ces préférences ordonnées, il faudrait privilégier un modèle d'encodage graphique où l'on utiliserait par exemple des lignes orientées depuis les champs des vecteurs opérandes vers le champ du vecteur résultat que ces champs de vecteurs opérandes servent à calculer. Au début, cela semblait être une approche très intéressante. Mais à l'essai, cette approche s'est avérée plutôt décevante. Par exemple, la fonction *intrinsic* $\_mm\_shuffle\_epi32$ permute les composants de son premier paramètre en fonction des composants de son second paramètre. Donc si nous avons $a = (A_3, A_2, A_1, A_0)$; $imm8 = (B_3, B_2, B_1, B_0)$; $r = (C_3, C_2, C_1, C_0)$ et $\_mm\_shuffle\_epi32(a, imm8) = r$; alors entre les composantes des trois vecteurs a, $imm8$ et r on a la relation $C_i = A_{B_i}$ telle qu'elle est sur la Figure 3.2. Comment indiquer avec une ligne que $C_0 = A_{B_0}$? En fait, après plusieurs tests et en suivant les recommandations appropriées en termes de conformité des couleurs (Levkowitz, 1997; MacDonald, 1999), mais aussi en remarquant la couleur utilisée par les ingénieurs d'Intel® lors de l'illustration graphique d'exemples particuliers (Stupachenko, 2015), nous choisissons d'assurer le lien avec une similarité de couleur. Plus important encore, nous concevons un formalisme de représentation visuelle de certaines opérations qui facilite à la fois la capture de la pensée de l'expert et son rendu visuel. Dans ce formalisme de représentation visuelle, par exemple $Inv\ X\ =\ 1-X$ et $A\ Idx\ B\ =\ A_B$. Ces représentations peuvent sembler

être du texte à première vue, mais ce sont bien des représentations visuelles (Engelhardt et Richards, 2020; MacDonald, 1999; Manovich, 2011; Strobelt *et al.*, 2015; Nualart-Vilaplana *et al.*, 2014) et ce sont des techniques de visualisation de logiciel (Myers, 1990). Mieux encore, comme Peirce et Bucher (1955) l'ont suggéré et illustré en ce qui concerne les équations algébriques de façon générale, il s'agit de représentations iconiques puisque le sens convoyé se retrouve dans l'exhibition des relations entre les termes utilisés. Pour construire graphiquement la structure *LinkingIndex* que l'expert remplit dynamiquement, nous utilisons une heuristique consistant à analyser le texte qui accompagne et explique le pseudo-code mis à disposition par le constructeur Intel®. Mais comme le résultat n'est pas toujours parfait, l'expert a la possibilité de modifier les opérandes et le résultat grâce aux opérations d'insertion et de suppression du menu du deuxième quadrant sur la Figure 3.3. Lors d'une insertion, l'expert peut spécifier le nombre de champs du vecteur à insérer, qui est un entier sous la forme 2^n ; n allant de 0 à 10. En suivant les normes pertinentes (Levkowitz, 1997; MacDonald, 1999), nous choisissons les couleurs pour atteindre un équilibre entre plusieurs objectifs. Le premier objectif est d'assurer un lien visuel entre le deuxième et le troisième quadrant. Le troisième quadrant contient, en plus des opérateurs, la définition du vecteur résultat et des vecteurs opérandes de la fonction vectorielle sélectionnées dans le premier quadrant. Le second quadrant comporte deux modes : le mode expert et le mode novice. En mode expert, le menu de suppression et d'insertion d'un vecteur opérande ou du vecteur résultat est affiché, ainsi que la description d'un champ du vecteur résultat par les champs des vecteurs opérandes utilisés pour calculer ce champ résultat. Le champ du vecteur résultat dont la définition est affichée et modifiable graphiquement est défini dans le troisième quadrant. En mode novice, la description de tous les champs du vecteur résultats déjà expliqués est affichée. Le deuxième objectif est de minimiser l'effet Stroop c'est-à-dire une distorsion entre la tâche sur laquelle l'attention doit être portée et un ou plusieurs

facteurs liés à cette tâche (Levkowitz, 1997; Algom et Chajut, 2019). Ainsi, nous veillons à ce que l'attention de l'utilisateur ne soit pas distraite. Dans notre cas, la tâche la plus importante, le but même de SIMDGiraffe, est de comprendre la description des champs du vecteur résultat. Le troisième objectif lors du choix des couleurs est d'éviter les images rémanentes lorsque l'utilisateur arrête de regarder l'écran, car cela entraîne un stress visuel dû à un visionnage prolongé (MacDonald, 1999). Il convient de noter que les deux premiers objectifs concernent davantage l'utilité du système alors que le troisième objectif concerne davantage l'utilisabilité ou l'ergonomie du système. Sur le plan ergonomique, nous respectons les principes de mise en œuvre d'une interaction homme-machine (St. Amant, 1998; Norman et Draper, 1986). Par exemple, l'affichage des boutons et des menus se fait en minimisant les étapes entre leur nécessité d'utilisation et la possibilité de leur utilisation ; en fait, il n'y a aucune étape. Nous choisissons des menus déroulants et dans la mesure du possible nous utilisons des champs pré-remplis. Nous séparons la vue expert de la vue novice puisqu'elles correspondent à deux modes d'utilisation différents du système (Norman et Draper, 1986). La vue novice disponible en mode novice est une vue de restitution ; elle restitue l'explication de l'expert. La vue expert disponible en mode expert est plus interactive ; elle structure et capture l'explication de l'expert lorsqu'il interagit avec le système.

3.2.1.4 Structuration, capture interactive et visualisation de la pensée de l'expert

La table *LinkingIndex* est remplie lorsque l'expert interagit graphiquement avec le système. Le vocabulaire dont dispose l'expert dans cette interaction est l'ensemble des opérateurs arithmétiques scalaires élémentaires augmentés d'opérateurs permettant de représenter des opérations non disponibles dans l'arithmétique élémentaire, mais essentielles dans notre contexte. À ce stade, les opérateurs augmentés sont Idx et Inv. Les opérateurs de l'arithmétique élémentaire sont

l'addition, la soustraction, la multiplication, la division euclidienne, la division de nombres réels. Bien que cette liste soit suffisante pour décrire la plupart des fonctions réelles, elle peut facilement être étendue si le besoin s'en fait sentir ; il suffit pour cela d'enregistrer le nouvel opérateur dans le fichier des opérateurs. Les parenthèses ne sont pas des opérateurs réels, bien qu'elles soient incluses dans la liste. Ces parenthèses sont utilisées par l'expert pour structurer sa démarche en fixant l'ordre de priorité de certaines opérations. L'esprit de l'expert est donc contraint par le vocabulaire visuel strict et les actions possibles avec les mots de ce vocabulaire. Mais la rigueur de ce vocabulaire possède, contrairement à ce que l'on pourrait penser d'un langage visuel qui est généralement contextuel (Marriott et Meyer, 1998; Futrelle, 1999; Erwig *et al.*, 2017), une sémantique universelle et sans équivoque. Cette sémantique universelle et cette clarté découlent du fait que les opérateurs arithmétiques utilisés font partie intégrante du langage mathématique qui a un sens clair et universel (Murphy, 2009; Marriott et Meyer, 1998). Ce vocabulaire de l'arithmétique élémentaire facilite l'expression de l'expert. Il est possible pour l'expert de modifier à tout moment la définition d'un champ d'un vecteur résultat. Pendant une session, toutes les traces laissées par l'expert sont persistantes. Ces traces sont ensuite utilisées pour générer une explication visuelle à laquelle le novice a accès dans la vue novice. Nous illustrons par deux cas d'utilisation comment SIMDGiraffe capture les explications de l'expert et les met à la disposition du novice. Les deux cas d'utilisation concernent `_mm512_mask_add_pd` et `_mm_shuffle_epi32`.

3.2.1.5 Étude de cas : `_mm512_mask_add_pd` et `_mm_shuffle_epi32`

i) `_mm512_mask_add_pd`

Nous utilisons dans SIMDGiraffe l'explication fournie par le constructeur Intel®

sur son site web [4] et qui consiste en un pseudocode accompagné d'un texte explicatif. L'expert assure le lien entre chaque champ du vecteur résultat et les champs d'opérandes utilisés dans le calcul de ce champ du vecteur résultat par le biais de la vue expert de SIMDGiraffe. Cette vue experte correspond au deuxième quadrant de la Figure 3.3. L'expert explique un champ du vecteur résultat à la fois. Notez que cette possibilité d'expliquer un champ du vecteur résultat à la fois est basée sur notre hypothèse résumée par l'équation 3.4. Pour cette explication, l'expert utilise l'opérateur Inv. Le novice a accès à la représentation visuelle des champs déjà expliqués par l'expert dans la vue novice de la Figure 3.4. Il y accède en cliquant sur le bouton « *view how to compute all fields* ». Dans le cas de la fonction `_mm512_mask_add_pd`, la première chose qui ressort est la régularité visuelle des champs ; c'est d'ailleurs le cas pour la majorité, voire la totalité, des instructions vectorielles. La formule

$$E_i \ = \ (1 - B_i)xA_i \ + \ B_ix(C_i + D_i) \tag{3.8}$$

émerge facilement à l'observation pour quiconque a maîtrisé les fonctions arithmétiques élémentaires. Bien entendu, dans cette formule, B_i vaut toujours 1 ou 0. La formule de l'équation 3.8 est donc beaucoup plus générale, mais beaucoup plus claire et précise ; surtout elle dispense le novice de faire un effort de conceptualisation et de généralisation à partir d'exemples comme dans certaines explications particulières, que ce soit celles d'Intel® (Stupachenko, 2015) ou d'autres (Van Hoey, 2019; Kusswurm, 2018; Dirty hands coding, 2019; Muła et Lemire, 2018). De plus, il est toujours plus facile d'appliquer une formule à des exemples particuliers, par exemple en faisant ici $B_i = 0$ ou $B_i = 1$, que de généraliser des exemples particuliers pour obtenir une formule.

---

4. Lien vers la présentation de l'*intrinsic* `_mm512_mask_add_pd` : `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#!=undefined&text=_mm512_mask_add_pd&ig_expand=7101,141`.

ii) `_mm_shuffle_epi32`

Pour ce cas d'utilisation, l'expert utilise l'opérateur Idx. Le pseudo-code et le texte explicatif disponibles sur le site du fabricant Intel® [5] et intégrés dans SIMDGiraffe sont présentés comme indiqué sur le premier quadrant de la Figure 3.2. D'autres ont consacré un chapitre entier de leur livre à l'explication de ce cas d'utilisation (Van Hoey, 2019). Les valeurs des champs du vecteur a qui est de type $\_\_m128i$ sont prélevées en fonction des valeurs des champs du vecteur masque $imm8$ qui est de type int. Le résultat est stocké dans le vecteur r qui est de type $\_\_m128i$ comme on peut le voir sur le troisième quadrant de la Figure 3.2. Tout novice qui maîtrise la notion algébrique élémentaire de tableau ou d'index comprend le lien entre chaque champ du vecteur résultat et les champs correspondants des vecteurs opérandes. Là encore, comme on peut le voir sur le deuxième quadrant de la Figure 3.2, la régularité visuelle facilite la généralisation. Le mélange inversé est un cas particulier où $B_3 = 00 = 0$; $B_2 = 01 = 1$, $B_1 = 10 = 2$, $B_0 = 11 = 3$, et donc $B = imm8 = 27$.

En principe, et suivant la méthodologie adoptée et la pratique en la matière (Sedlmair *et al.*, 2012; Munzner, 2008), les études de cas tiennent en général lieu de validation. Mais au-delà de cette procédure de validation traditionnelle pratiquée et acceptée en InfoViz et en SV, nous conduisons avec ces deux *intrinsics* et leurs vues novices respectives générées par le module 1 de SIMDGiraffe une étude expérimentale randomisée afin de tester de manière holistique la SV en tant que science. Quant à la vue expert, son évaluation et sa validation peuvent se faire par le nombre de fonctions vectorielles supportées, et par la possibilité pour l'expert de modifier la représentation graphique des fonctions vectorielles

---

5. Lien vers la présentation de l'*intrinsic* `_mm_shuffle_epi32` : `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#!=undefined&text=_mm_shuffle_epi32&ig_expand=150,6088`.

par le menu d'insertion et de suppression. Si nous disposons du fichier de description d'un fabricant, comme celui du fabricant Intel® librement disponible en ligne, nous pouvons produire la représentation graphique des fonctions vectorielles incluses dans ce fichier en utilisant SIMDGiraffe. Le pourcentage des fonctions vectorielles couvertes peut donc être considéré comme proche de 100%. Du point de vue ergonomique, nous suivons les règles relatives aux interfaces et interactions humain-machine (HCI pour *Human-Computer Interaction* en anglais) (Norman et Draper, 1986) dans la conception des menus. Ces possibilités d'interaction sont illustrées par les captures d'écran produites par SIMDGiraffe, à savoir la Figure 3.3 pour la fonction vectorielle `_mm512_mask_add_pd` et la Figure 3.2 pour la fonction vectorielle `_mm_shuffle_epi32`. Des images GIF animées où un expert interagit avec SIMDGiraffe pour générer les représentations visuelles de ces deux fonctions vectorielles sont également permanemment disponibles sur *Open Science Framework* (OSF) à l'adresse `https://osf.io/bgwdm/?view_only=1daefc7ed7af40809ab906c752c19409` et sur GitHub à l'adresse `https://github.com/pmntang/SIMDGiraffe`. Le prototype peut être testé en ligne à l'adresse `https://pmntang.github.io/SIMDGiraffe`. Le module 2 quant à lui vise à aider à comprendre le comportement d'un code vectoriel, c'est-à-dire un code créé en utilisant des fonctions vectorielles (*intrinsics*) ou même leurs équivalents en assembleur.

3.2.2 Conception et mise en œuvre d'un outil de visualisation afin de supporter la compréhension de comportement de codes vectoriels [6]

En nous appuyant sur le modèle de comportement d'un code qui s'exécute sur une architecture cible donnée (sous-section 3.1.2, page 63), nous pouvons maintenant

6. Les idées développées dans cette partie font l'objet d'un article publié intitulé « SIMDGiraffe : Visualizing SIMD Functions », et présenté à IVAPP 2021.

spécifier le problème à résoudre. Il s'agit d'aider un programmeur à comprendre le comportement d'un code vectoriel s'exécutant sur une architecture donnée. Nous nous intéressons aux codes de l'ordre de dizaine de lignes c'est-à-dire typiquement les codes de fonctions.

3.2.2.1 Spécification du problème, identification des données, abstraction des opérations et type abstrait de données (TAD) à utiliser

Étant donné un code source S produit à l'aide d'*intrinsics* ou de leurs équivalents en assembleur, comprendre le comportement de S lors de son exécution sur une architecture vectorielle H revient à comprendre les interactions entre les cellules de la matrice PE résultant du calcul $F(S, H)$ comme l'indique l'équation 3.6. D'après notre modèle, chaque cellule de cette matrice décrit l'interaction entre l'équivalent assembleur d'une instruction du code S et les registres ou la mémoire que cette instruction utilise. Afin d'obtenir les données à utiliser pour la visualisation, il nous faut effectivement calculer la matrice PE. Une solution pourrait être l'instrumentation du code S ; mais nous savons que cette solution est susceptible de modifier le comportement que nous souhaitons observer (Diehl, 2007). Des outils comme le LLVM Machine Code Analyzer (LLVM, 2018) embarqué dans l'application web Godbolt (Godbolt, 2020) nous permettent de calculer la sortie de la fonction F. En effet, cet outil permet de générer, en fonction de l'architecture cible passée en paramètre, des données décrivant le comportement d'un code qui est exécuté sur cette architecture. Ainsi, SIMDGiraffe s'appuie sur Godbolt pour la génération de ces données, qu'il récupère et traite. Mais avant de passer au traitement informatique des données dans SIMDGiraffe, il faut conceptuellement abstraire de ces données le comportement du code en question. Cette abstraction est concrétisée par un type abstrait de données, qui structure les possibilités en termes d'encodage visuel et d'interactions. Les données récupérées de Godbolt subissent donc une transformation logique et formelle pour s'adapter au

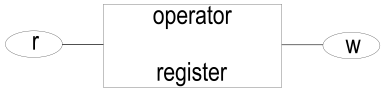


FIGURE 3.5 – Représentation du triplet lecture, opération, écriture comme une séquence graphique.

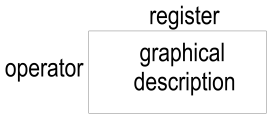


FIGURE 3.6 – Séquence graphique transformée en une cellule géométrique.

type abstrait de données qu'elles alimentent. Cette transformation est faite dans SIMDGiraffe. Il s'agit principalement d'épurer les données reçues pour n'en retenir que les plus pertinentes, mais aussi de les transformer conformément au modèle développé. D'après ce modèle, il apparaît que le TAD, et par suite la structure de données qui convient le mieux pour la représentation au sens de Spence (2014) est une matrice puisque le modèle lui-même repose sur une structure matricielle.

3.2.2.2 Encodage visuel et conception des interactions

De la manière dont la matrice PE est obtenue, l'encodage, et les interactions qui suivent traduisent le comportement du code lorsqu'il est exécuté sur l'architecture cible. Pour trouver l'encodage visuel qui traduit la description portée par la matrice d'une manière cognitivement efficace, nous suivons les principes de l'encodage graphique (Engelhardt et Richards, 2020) et les règles de la palette de couleurs (MacDonald, 1999). L'assortiment des couleurs est réalisé lors de l'implémentation car le support d'affichage doit être pris en compte. Cet assortiment se fait par l'application de ces règles et principes, mais aussi par des ajustements en fonction du rendu visuel obtenu. Dans la description portée par la matrice,

nous ne considérons que les entrées/sorties et les opérations sur les registres, car elles reflètent le comportement du code. Pour le code machine, nous assimilons ce comportement à sa sémantique. En termes de sémantique, une instruction-machine peut être modélisée en trois étapes simples : lire les opérandes sources, effectuer une opération, écrire sur les opérandes de destination (Dasgupta *et al.*, 2019). Ainsi, nous pouvons décrire complètement un programme par une séquence de triplets lecture, opération, écriture. Chaque élément d'un tel triplet correspond à la modification de l'état d'un ou plusieurs registres. Nous pouvons traduire ce triplet par la séquence graphique de la Figure 3.5. Dans cette représentation, r correspond à la lecture, et w à l'écriture. On suppose que s'il n'y a pas de lecture ou d'écriture pendant une opération sur un registre, l'ellipse correspondante est laissée vide. Nous modifions légèrement cette figure dans la représentation visuelle pour obtenir la Figure 3.6. Pour la plupart des taxonomies de visualisation logicielle (Diehl, 2007; Myers, 1990; Price *et al.*, 1998) la visualisation du comportement du code à l'exécution implique une visualisation dynamique même s'il est établi qu'une visualisation statique, si elle peut fournir les mêmes informations qu'une visualisation dynamique, est meilleure en termes d'efficience cognitive et d'efficacité dans la compréhension (Robertson *et al.*, 2008). Pour décrire complètement l'action d'une opération sur un registre, nous n'avons pas besoin d'informations supplémentaires de la part de l'utilisateur ou de toute autre entité ; nous avons seulement besoin du code source et de l'architecture cible. Cette action n'est pas non plus dépendante du temps. En admettant que pour une instance et un registre (ou toute autre mémoire) donnés toutes les informations à visualiser puissent être affichées dans un rectangle similaire à celui de la Figure 3.6, le choix d'une visualisation statique est approprié pour présenter un schéma global du comportement du code en question. La matrice PE est donc traduite en une série d'images décrivant le comportement du code dans un plan et les noms des lignes et des colonnes sont ajoutés.

Mais, cette présentation statique est insuffisante pour rendre compte des interactions entre les différentes parties du code. L'interactivité du rendu graphique est conçue pour corriger cette insuffisance. Les objectifs de l'interactivité de ce rendu graphique sont de permettre à l'utilisateur de découper le code source vectoriel en blocs logiques en fonction du comportement de ce code lors de son exécution, de localiser dans le code source les instances d'une instruction vectorielle apparaissant sur la représentation visuelle, d'avoir des explications sur chacune des instructions vectorielles apparaissant dans la décomposition du code source vectoriel. Cette interactivité repose en grande partie sur des relations formelles qui relient les éléments entre eux. Ainsi en reprenant notre modèle, on peut définir la relation G sur R par $(i_n, r_m)\ G\ (i_k, r_l)$ si et seulement si l'entrée du registre r_l pour l'instruction i_k est lue sur le registre r_m et écrit (sur r_m) par l'instruction i_n ou alors $(n, m) = (k, l)$. Nous considérons que pour utiliser un registre, une instruction lit une valeur pour ce registre, et à la fin de son exécution l'instruction écrit la valeur résultante dans le registre. Deux cas de figure peuvent alors se présenter : *i)* Si l'opération portée par l'instruction a pour conséquence la modification de la valeur lue, alors l'instruction aura modifié la valeur du registre ou plus simplement le registre ; *ii)* si l'opération portée par l'instruction ne modifie pas la valeur lue, alors tout se passe comme si l'instruction lit une valeur pour le registre et à la fin écrit la même valeur sur le registre. Si on pose $P_{x_0} = \{x \in R\ et\ \exists y \in R \mid (y\ G\ x_0\ ou\ x_0\ G\ y)\ et\ (y\ G\ x\ ou\ x\ G\ y)\}$, on définit à partir d'un point donné x_0 de R un chemin partant du point d'entrée du code S vu comme une fonction, jusqu'à un point de sortie de cette fonction. Un point de sortie est défini ici comme un point où la fonction accède à la mémoire pour y écrire sans que la valeur ainsi écrite ne soit plus accédée pendant son exécution. Un tel parcours permet d'isoler des blocs logiques de code indépendants ou faiblement dépendants. Un bloc est indépendant s'il peut s'exécuter indépendamment du reste de la fonction jusqu'à son point de sortie. Un bloc est faiblement

dépendant s'il peut s'exécuter indépendamment du reste de la fonction, mais sa valeur de retour est lue en interne dans la fonction. Dans SIMDGiraffe, il suffit à l'utilisateur de sélectionner x_0 et de voir P_{x_0}. Jusqu'à deux points peuvent être sélectionnés et les chemins correspondants visualisés en même temps. Un point est sélectionné soit en le pointant avec la souris et dans ce cas, il cesse d'être un point sélectionné lorsqu'il n'est plus pointé, soit en cliquant dessus et dans ce cas, il est sélectionné jusqu'à ce qu'on clique à nouveau dessus. Lorsqu'un point est sélectionné sur le graphique, l'instance de l'instruction vectorielle correspondante dans la fenêtre graphique et le bloc de code correspondant dans le code source sont sélectionnés. La sélection dans le code source est matérialisée par une mise en surbrillance. Il existe donc une correspondance visuelle interactive entre le code source et la représentation visuelle de son comportement lorsqu'il est exécuté sur l'architecture cible. Une explication de l'instruction vectorielle dont une instance est sélectionnée est également donnée dans l'espace de représentation graphique. Bien que certains échantillons de code soient préchargés dans SIMDGiraffe, l'utilisateur peut saisir son propre code source vectoriel et visualiser interactivement son comportement lorsqu'il est exécuté sur l'architecture cible.

3.2.2.3 Test et validation du module 1 de SIMDGIraffe et du modèle sous-jacent : deux cas d'utilisation

Conformément à la méthodologie suivie (Munzner, 2009), afin de tester et valider ce module du prototype SIMDGiraffe, mais aussi pour confirmer (ou réfuter) la consistance du modèle ; afin d'apporter des éléments de réponse à la question de recherche (2.5, page 49) et confirmer la consistance des hypothèses sous-jacentes (2.5, page 49), nous présentons deux cas d'utilisation. Dans ces deux cas d'utilisation, l'architecture vectorielle cible, qui est paramétrée dans le code source de SIMDGiraffe, est l'AVX-512 d'Intel®. Le cas d'utilisation de la Figure 3.7 montre la visualisation de la fonction `interleave_uint8_with_zeros_avx_lut`. Dans la

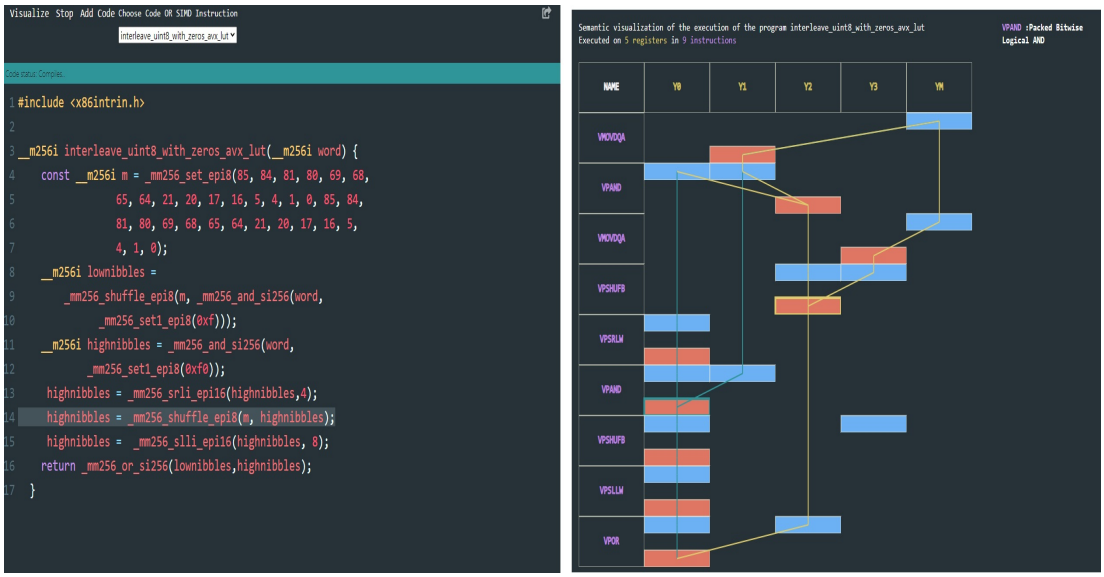


FIGURE 3.7 – Visualisation du comportement du programme `interleave_uint8_with_zeros_avx_lut` lors de son exécution sur une l'architecture SIMD de l'AVX512.

fenêtre de droite, il y a un résumé de l'exécution de la fonction sur l'architecture cible au-dessus du plan de visualisation. Ainsi, sur l'AVX-512, cette fonction est exécutée avec 5 registres et en neuf (9) instances d'instructions vectorielles. À droite, en haut de cette fenêtre, nous avons l'explication de l'instruction vectorielle VPAND, dont une des instances est la dernière instance sélectionnée. Les cellules de la matrice sont représentées par des rectangles et découlent de la Figure 3.6. La première cellule sélectionnée a un bord de rectangle jaune, et la deuxième cellule sélectionnée a un bord de rectangle vert. Dans la fenêtre de gauche, l'instruction en surbrillance, c'est-à-dire la ligne 14, correspond à la dernière instance sélectionnée, c'est-à-dire la deuxième instance de l'instruction VPAND ou encore la sixième entrée dans le tableau avec entête de la représentation graphique. Le chemin en jaune matérialise un bloc de code qui est faiblement dépendant du reste du code.

Dans le cas d'utilisation de la Figure 3.8, nous avons également deux instances

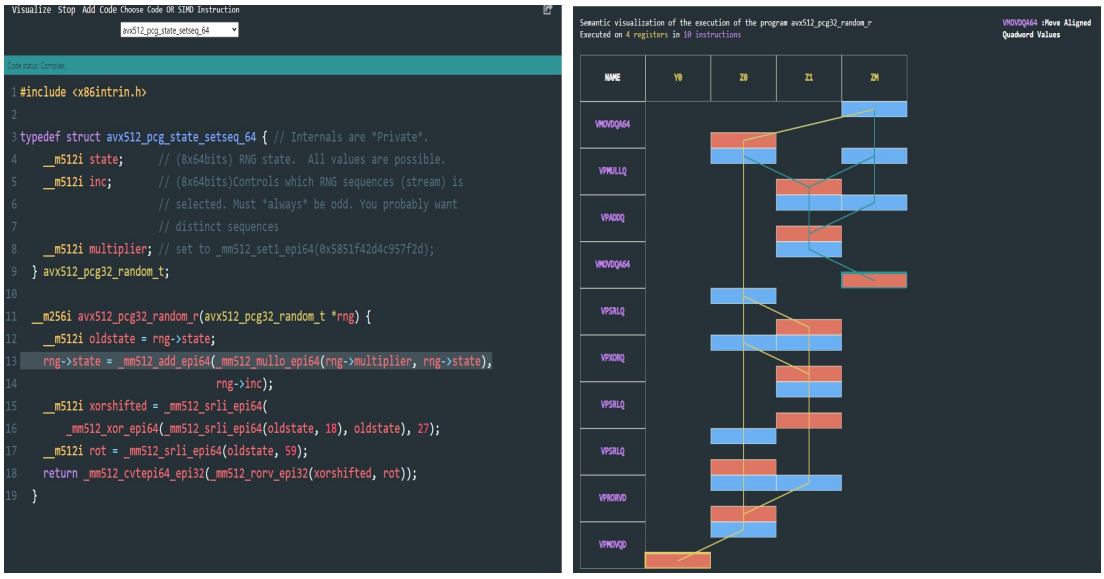


FIGURE 3.8 – Visualisation du comportment du programme `avx512_pcg_state_setseq_64` lors de son exécution sur une l'architecture SIMD de l'AVX512.

sélectionnées. Ce que l'on peut remarquer, c'est que les deux cellules délimitent par leurs chemins respectifs deux blocs de codes indépendants. La fonction correspondant à ce code source accède et écrit pendant son exécution en mémoire au niveau du premier bloc de code, comme on peut le voir. L'observation du comportement de cette fonction nous montre que nous pouvons diviser le code source en deux grands blocs qui partagent une seule déclaration et affectation de variable à l'entrée de chacun des deux blocs. Voici l'instruction d'affectation en question dans le code source :

```
__m512i oldstate = rng->state;
```

Le premier bloc de représentation graphique matérialisant le comportement de code pendant l'exécution, et dont le chemin dans la représentation visuelle est en bleu, correspond au premier bloc du code source :

```
        rng->state=_mm512_add_epi64(_mm512_mullo_epi64(rng->multiplier,
rng->state), rng->inc);
```

Le deuxième bloc, donc le chemin est en jaune, correspond au deuxième bloc du code source :

```
        __m512i xorshifted = _mm512_srli_epi64(_mm512_xor_epi64(
_mm512_srli_epi64(oldstate, 18), oldstate), 27);
        __m512i rot = _mm512_srli_epi64(oldstate, 59);
        return _mm512_cvtepi64_epi32(_mm512_rorv_epi32(xorshifted, rot));
```

La détermination du début et de la fin de chaque bloc dans le code source se fait en fixant respectivement le point d'entrée et de sortie dans la représentation visuelle. Le début et la fin du bloc sont alors respectivement mis en évidence chaque fois. Ces deux blocs fonctionnent indépendamment. Grâce à la représentation visuelle, une personne peu expérimentée en programmation vectorielle a pu réaliser ce découpage, tout comme elle a pu constater l'accès en écriture en mémoire par la fonction.

Nous venons de présenter dans ce chapitre deux modèles que nous avons développés. Le premier de ces modèles relève du domaine des fonctions vectorielles (ou *intrinsics*) et le second relève du domaine du comportement de code (en particulier de code développé en utilisant des *intrinsics*). Nous avons aussi présenté comment partant de ces modèles, nous avons conçu et implémenter les modules respectifs correspondants de SIMDGiraffe. Du point de vue de l'InfoViz ou de la SV, les cas d'utilisation que nous avons présentés sont suffisants en termes de validation de ces modules (Munzner, 2008; Chotisarn *et al.*, 2020). Mais du point de vue de notre question de recherche, nous avons abordé essentiellement des aspects pratiques, même si ce faisant nous avons apporté plusieurs innovations comme le

fait d'utiliser un outil de visualisation pour capturer l'expertise de chez l'expert, comme les modèles que nous avons développés, ou même comme l'élargissement de la pratique de la SV au domaine de la programmation vectorielle. Dans le chapitre suivant, nous allons aborder des aspects beaucoup plus théoriques de cette question de recherche, notamment en essayant de falsifier la SV.

CHAPITRE IV

CORROBORATION DE LA SV PAR L'EXPÉRIMENTATION : MÉTHODOLOGIE, DONNÉES RÉCOLTÉES, ANALYSES, RÉSULTATS ET DISCUSSION

Puisque nous avons suivi les prescriptions de la SV en particulier et de l'InfoViz en général dans la conception et l'implémentation du prototype SIMDGiraffe, ce prototype peut servir comme un potentiel falsificateur de la SV. En clair, SIMDGiraffe peut contribuer à trancher la question de savoir si oui ou non la visualisation aide à la compréhension du comportement de code, et plus exactement du code vectoriel. Mais avant la réponse à la question proprement dite, nous présentons d'abord dans la section 4.1 les outils mobilisés pour arriver à cette réponse. Ensuite dans la section 4.3 nous présentons les données récoltées à l'issue de l'administration du questionnaire en ligne. Puis à la section 4.4 nous explorons ces données et examinons les conclusions qui en découlent. Enfin dans la section 4.5 nous mettons en perspective les résultats obtenus. Toutes les données mentionnées (brutes et calculées) sont disponibles et librement accessibles sur OSF à l'adresse `https://osf.io/bgwdm/?view_only=1daefc7ed7af40809ab906c752c19409` et sur le site du projet SIMDGiraffe à l'adresse `https://github.com/pmntang/SIM`

`DGiraffe/tree/master/survey_materials` [1].

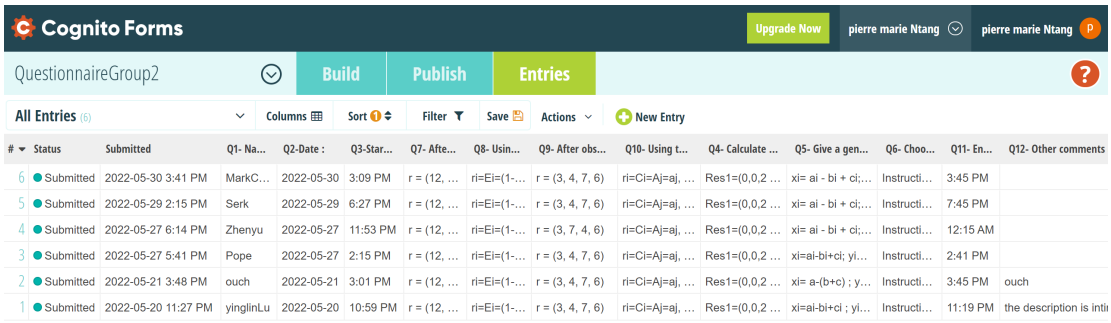


FIGURE 4.1 – Vue sur https://www.cognitoforms.com/ des individus ayant rempli la version en anglais du questionnaire de type 2 (Group 2).

4.1 Méthodologie

Aucun énoncé reposant sur une quantification universelle n'est expérimentalement vérifiable (Popper, 2005). Ainsi, en raison de son caractère universel, l'affirmation selon laquelle la SV facilite la compréhension du comportement de code ne peut pas être vérifiée. Mais nous pouvons corroborer par un observable construit en suivant les hypothèses de la SV en tant que science que les prédictions de cette science se réalisent, c'est-à-dire que nous pouvons instancier cette affirmation universelle sur une observation concrète. Disposant donc du prototype SIMDGiraffe, la question instanciée est la suivante : ce prototype facilite-t-il réellement la compréhension du comportement d'une fonction vectorielle ou *intrinsic* ? Autrement dit, étant donné un *intrinsic* et deux personnes, y a-t-il une différence, toute autre chose égale par ailleurs, entre les deux personnes dans la compréhension du comportement de cet *intrinsic* si l'une d'elles utilise SIMDGiraffe et l'autre

1. Les idées développées dans ce chapitre font l'objet d'un article à venir qui est intitulé : « SIMDGiraffe : Capturing and Visualizing the Expert's Mind to Understand SIMD Functions ».

TABLEAU 4.1 – Formule de calcul du score de performance d'une personne à partir des tâches effectuées.

intrinsic	Question liée à la tâche	Score de performance pour la tâche	Score de performance pour l'*intrinsic*	Score final de la personne
_mm512_mask_add_pd	Q7	$s_1^1 = \begin{cases} 1 & si\ vrai \\ 0 & si\ faux \end{cases}$	$s_1 = \frac{2}{3}(s_1^2 + \frac{1}{2}s_1^1)$	$s = \frac{1}{2}(s_1 + s_2)$
	Q8	$s_1^2 = \begin{cases} 1 & si\ vrai \\ 0 & si\ faux \end{cases}$		
_mm_shuffle_epi32	Q9	$s_2^1 = \begin{cases} 1 & si\ vrai \\ 0 & si\ faux \end{cases}$	$s_2 = \frac{2}{3}(s_2^2 + \frac{1}{2}s_2^1)$	
	Q10	$s_2^2 = \begin{cases} 1 & si\ vrai \\ 0 & if\ faux \end{cases}$		

pas. Bien sûr, d'après la SV cette différence devrait se traduire par une meilleure compréhension du comportement de l'*intrinsic* par la personne qui utilise SIMDGiraffe comparativement à celle qui ne l'utilise pas. Si nous montrons qu'il y a une différence de compréhension du comportement de code en faveur de la personne qui utilise SIMDGiraffe, la SV s'en trouve empiriquement renforcé du fait du risque de falsification encouru. Mais puisqu'il est pratiquement impossible de trouver deux personnes toute chose égale, nous recourons aux outils statistiques pour répondre à la question. Ainsi, afin d'éviter tout biais, nous construisons de manière rigoureusement aléatoire des groupes auxquels nous assignons des tâches permettant de mesurer la performance dans la compréhension du comportement des *intrinsics*. Les tâches sont exécutées dans les mêmes conditions, excepté le fait que parmi ces groupes, il y en a un dont les individus n'utilisent pas SIMDGiraffe dans l'exécution des tâches qui leur sont soumises. Nous comparons ensuite les performances des différents groupes. De cette façon, compte tenu de la distribution rigoureusement aléatoire des individus, il n'est pas possible qu'un caractère, même inconnu, se retrouve dans un seul groupe, et les biais dus aux différences individuelles sont corrigés. Avant de procéder à la comparaison de la performance des différents groupes dans l'exécution des tâches assignées, il importe de spécifier ces tâches et de détailler la métrique des performances.

4.1.1 Spécification des tâches exécutées et de la métrique utilisée pour mesurer la performance

Rappelons que comprendre le comportement d'un *intrinsic*, c'est être capable de déterminer la relation entre les entités x, y d'une part et l'entité z d'autre part dans l'équation 3.7 (3.2.1.1, page 68). Étant donnés une personne participante et un *intrinsic* à l'étude, nous pouvons distinguer en particulier deux tâches que la personne doit faire au regard de l'*intrinsic* :

i) La première tâche consiste à essayer de comprendre comment est calculé z à partir de x et de y ; c'est-à-dire de déterminer la formule permettant d'effectuer ce calcul. Pour faciliter l'exécution de cette tâche, et en tirant les enseignements lors de la phase de test dans notre démarche, nous soumettons à la personne quatre formules parmi lesquelles elle doit choisir la bonne réponse. Le score s_i^j ($avec\ j = 1\ et\ i \in \{1, 2\}$) de sa performance pour la tâche est alors de 1 point si la réponse est bonne (vrai) et de 0 point dans le cas contraire (faux) comme le montre le Tableau 4.1. Nous assimilons cette tâche à la maîtrise de connaissance relativement au comportement de l'*intrinsic* ; la personne qui effectue cette tâche avec succès est considérée comme connaître le comportement de l'*intrinsic*, c'est-à-dire la formule liant ses opérandes et son résultat.

ii) La seconde tâche consiste à essayer d'utiliser la formule permettant de calculer z à partir de x et de y. Là aussi et suivant les mêmes raisons, nous soumettons à la personne une application de la formule en question sur un exemple concret avec quatre propositions de résultats. Le score s_i^j ($avec\ j = 2\ et\ i \in \{1, 2\}$) de sa performance pour la tâche est alors de 1 point si la réponse est bonne (vrai) et de 0 point dans le cas contraire (faux) comme le montre le Tableau 4.1. Nous assimilons cette tâche à la maîtrise de la compétence relativement à l'application de la compréhension du comportement de l'*intrinsic* ; la personne

qui effectue cette tâche avec succès est considérée comme savoir mettre en pratique sa connaissance du comportement de l'*intrinsic*.

Le score s_i ($avec\ i \in \{1, 2\}$) de performance pour l'*intrinsic* de la personne pour l'exécution des deux tâches est obtenu en combinant les deux scores s_i^j ($avec\, i,\ j\ \in \{1, 2\}$) comme le montre le Tableau 4.1. Ce score varie entre 0 et 1. Nous faisons le choix à travers la formule de calcul de ce score d'accorder plus d'importance à la compétence par rapport à la connaissance en ce qui concerne la compréhension du comportement de l'*intrinsic* ; ce qui se traduit par une plus grande pondération de la compétence. Nous faisons exécuter ces deux tâches par chaque personne participante pour deux *intrinsics*, à savoir `_mm512_mask_add_pd` et `_mm_shuffle_epi32`. Le score final de la personne $s = \frac{1}{2}(s_1 + s_2)$ est obtenu en combinant les scores relatifs à chacun des deux *intrinsics* comme le montre le Tableau 4.1. Ce score varie entre 0 et 1.

4.2 Mise en œuvre [2]

Nous élaborons ensuite les outils et la démarche appropriés pour recruter des personnes et leur faire exécuter ces tâches dans les conditions décrites. Parmi les documents utilisés et qui figurent en annexe, les lettres de sollicitation et les questionnaires ont une importance pratique capitale, puisque ce sont des documents « de terrain », donc utilisés pour le recrutement des personnes participantes et la collecte des données.

---

2. Tous les documents sont joints en annexe et disponibles sur le site web de l'OSF à l'adresse `https://osf.io/bgwdm/?view_only=1daefc7ed7af40809ab906c752c19409` ou sur le site web du projet SIMDGiraffe à l'adresse `https://github.com/pmntang/SIMDGiraffe/tree/master/survey_materials`.

4.2.1 Les lettres de sollicitation

Nous utilisons trois types de lettres de sollicitation dans cette étude. Chaque type de lettre est bilingue avec un premier texte en anglais et un second texte en français dans la même lettre de sollicitation. Dans le texte anglais, il y a un lien vers un questionnaire d'un certain type en anglais. Dans le texte français, il y a un lien vers un questionnaire d'un certain type en français. Pour le type de questionnaire dont le lien est contenu dans un type de lettre, les deux versions anglaise et française du questionnaire en question sont identiques et seule la langue utilisée dans chaque version du questionnaire diffère.

4.2.2 Les questionnaires

Nous utilisons trois types de questionnaires. Un questionnaire de type 1 contient pour chaque fonction vectorielle utilisée, un lien vers une vidéo sur YouTube. Dans cette vidéo, et en fonction de la langue du questionnaire, une explication est donnée par un expert dans le domaine des fonctions vectorielles. Pour un questionnaire en anglais, le lien pour la vidéo qui explique la fonction vectorielle `_mm512_mask_add_pd` est `https://youtu.be/aoAX922SeGs` ; et le lien pour la vidéo qui explique la fonction vectorielle `_mm_shuffle_epi32` est `https://www.youtube.com/watch?v=zSz_-eAl8MA`. Pour un questionnaire en français, le lien pour la vidéo qui explique la fonction vectorielle `_mm512_mask_add_pd` est `https://youtu.be/omQ0ebeYJBg` ; et le lien pour la vidéo qui explique la fonction vectorielle `_mm_shuffle_epi32` est `https://www.youtube.com/watch?v=WVz1jHTIOtY`. Il y a également, pour chaque fonction vectorielle utilisée, une capture d'écran de l'explication de cette fonction vectorielle. Cette capture d'écran est tirée du site Internet d'Intel®. Cette capture d'écran insérée dans un questionnaire permet aux participants de ne pas être obligés d'aller sur le site du fabricant, mais en même temps d'avoir l'explication telle qu'elle apparaît sur ce site. Les liens vers

les questionnaires de type i sont contenus dans la lettre de type i, $i \in \{1, 2, 3\}$. Un questionnaire de type 2 contient, pour chaque fonction vectorielle utilisée, une capture d'écran qui est la représentation graphique de l'explication de l'expert pour cette fonction. C'est l'interaction de l'expert avec SIMDGiraffe qui génère cette explication. Pour la fonction vectorielle `_mm512_mask_add_pd`, la capture d'écran ressemble à la Figure 3.3 mais où le second quadrant est remplacé par la Figure 3.4. Pour la fonction vectorielle `_mm_shuffle_epi32`, la capture d'écran ressemble à la Figure 3.2. Comme on peut le remarquer, ces captures d'écran correspondent à la vue novice de chacune de ces fonctions dans SIMDGiraffe. La seule différence entre un questionnaire de type 3 et un questionnaire de type 2 est qu'un questionnaire de type 3 contient des liens vers des vidéos sur YouTube. Ces liens sont exactement les mêmes que ceux contenus dans un questionnaire de type 1. Un questionnaire contient 12 questions [3]. Mais les questions concernant la date (Q2-Date) et l'heure (Q3-Heure de début) sont automatiquement remplies et verrouillées. Une question (Q11-Heure de Fin) concerne l'heure de fin et est préremplie par le terminal sur lequel le questionnaire est rempli, en fonction de l'horloge de ce terminal. Une question facultative (Q12-Autres commentaires et remarques) permet de faire des commentaires et remarques libres. Une question (Q5- Donnez une formule générale pour calculer les coordonnées de Res1(xi) et Res2(yi) en fonction de celles de A (ai), B (bi) et C (ci). xi= ? yi= ?) nécessite la saisie d'une réponse dans un champ de texte. Le reste des questions sont des questions à choix multiple avec une seule réponse correcte. Pour ces questions à choix multiples, le répondant est tenu de choisir une réponse, sinon le questionnaire ne peut être soumis. À l'exception de la question Q12, les autres questions sont

3. Voir les questionnaires en anglais joints en annexe pour la version en anglais des questions et des énoncés, qui sont la traduction en anglais des questions et des énoncés de la version en français.

obligatoires et le questionnaire ne peut être soumis si ces questions n'ont pas de réponse. Les questions utilisées dans le calcul du score d'une personne sont Q7, Q8, Q9 et Q10. Pour le questionnaire de type 1, les énoncés de ces questions sont les suivants :

i) Q7 : Après avoir lu la description et l'explication ci-dessus, dites ce que fait l'*intrinsic* en effectuant le calcul suivant : étant donné `src = (1, 3, 4, 1, 2, 5, 4, 1, 2, 3, 4, 1, 1, 3, 4, 1); k = (1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0); a = (6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1); b = (6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1)`. Calculer `r = _mm 512_mask_add_ps (src, k, a, b)` ;

ii) Q8 : En utilisant à nouveau la description et l'explication ci-dessus, donner une formule générale pour calculer les coordonnées de $r(r_i)$ en fonction de celles de $src(src_i)$, $k(k_i)$, $a(a_i)$ et de $b(b_i)$;

iii) Q9 : Après avoir lu la description et l'explication ci-dessus, dites ce que fait l'*intrinsic* en effectuant le calcul suivant : étant donné `a = (6, 7, 4, 3)` ; `imm8 = (0, 1, 2, 3)`. Calculer `r = _mm_shuffle_epi32(a, imm8)` ;

iv) Q10 : En utilisant à nouveau la description et l'explication ci-dessus, donnez une formule générale pour calculer les coordonnées de $r(r_i)$ en fonction de celles de $a(a_i)$ et $imm8(imm8_i)$.

De façon générale, les questions du questionnaire de type 2 et du questionnaire de type 3 sont fondamentalement les mêmes que celles du questionnaire de type 1, hormis le fait que l'expression « description et explication ci-dessus » est remplacée par l'expression « figure ci-dessus » et que le verbe lire (lire la description) est remplacé par le verbe observer (observer la figure). « La description et l'explication ci-dessus » ou « la figure ci-dessus » fait référence à la description ou à la figure qui est incluse dans le questionnaire correspondant. Tous les questionnaires sont remplis en ligne et une fois qu'ils sont soumis, il n'y a aucune possibilité de les

modifier.

4.2.3 Démarche utilisée pour recruter les participants à l'étude

La période de collecte de données s'étend du 12 mai 2022 au 30 juin 2022. Elle correspond au moment où nous envoyons les premiers courriels, et au moment où nous n'envoyons plus de courriels de rappel. Nous constituons d'abord une liste de diffusion à partir des adresses électroniques des étudiants inscrits au moment de la période de collecte à un cours de mathématiques ou d'informatique à l'Université TELUQ. Les adresses électroniques des abonnés au blog du professeur Daniel Lemire ainsi que des adresses électroniques provenant de diverses autres sources sont également ajoutées à cette liste de diffusion. Au total, la liste de diffusion comprend 4329 adresses électroniques. Ces adresses électroniques sont réparties aléatoirement en trois groupes, à savoir le groupe 1, le groupe 2 et le groupe 3. À la fin de la période d'étude, le groupe 1 et le groupe 2 contiennent chacun 1442 adresses électroniques et le groupe 3 contient 1443 adresses électroniques. Personne (même pas l'expérimentateur) ne sait à quel groupe appartient une adresse courriel donnée. Les personnes sollicitées ne savent pas qu'il y a plusieurs groupes, et le message de sollicitation est individualisé. Ce message de sollicitation fait partie de l'en-tête du message envoyé, et la lettre fait partie du corps du message ; nous avons donc évité d'envoyer la lettre en pièce jointe. La lettre de type 1 a été envoyée aux adresses électroniques du groupe 1 ; la lettre de type 2 a été envoyée aux adresses électroniques du groupe 2 ; et la lettre de type 3 a été envoyée aux adresses électroniques du groupe 3.

4.3 Données récoltées

Nous utilisons la plateforme `https://www.cognitoforms.com/` qui permet de créer un formulaire par type de questionnaires. Une fois les formulaire créés, on

TABLEAU 4.2 – Distribution des individus par groupes et par langue des questionnaires.

		Langue du questionnaire		Total de questionnaires ou d'individus par groupe n_i $i \in \{1,2,3,T\}$	Score moyen (mo_i) ou median (me_i) par groupe (ou par type de questionnaires ; $i \in \{1,2,3,T\}$) $mo_i = \frac{Somme\ des\ scores^1\ du\ groupe}{Total\ des\ individus\ du\ groupe}$
		Questionnaires en Anglais	Questionnaires en Français		
Type du questionnaire (ou Groupe des individus)	I	10	16	$n_1 = 26$	$mo_1 = 0.45$; $me_1 = 0.33$
	II	6	14	$n_2 = 20$	$mo_2 = 0.82$; $me_2 = 1.00$
	III	5	7	$n_3 = 12$	$mo_3 = 0.69$; $me_3 = 0.75$
Total de questionnaires (ou des individus)		21	37	$n_T = 58$	$mo_T = 0.63$; $me_T = 0.75$

[1] Il s'agit des scores du tableau 4.3

peut générer un lien vers le type de questionnaire en question. C'est ce lien qui est inclus dans les messages de sollicitation. Une fois les questionnaires remplis et soumis, on peut les exporter questionnaire par questionnaire sous format PDF par exemple, ce qui revient à les exporter individu par individu. On peut aussi les exporter par type de questionnaires sous forme de tableau avec en ligne les individus et en colonne les champs du formulaire. Pour chaque individu, un champ identifiant unique (`Questionnaire_id`), séquentiellement numéroté est automatiquement ajouté, ainsi que la date et l'heure de soumission du questionnaire ainsi qu'on peut le remarquer sur l'aperçu de la Figure 4.1. Le tableau 4.2 résume le nombre de participants par type de questionnaire et par langue ainsi que le score moyen (moyenne arithmétique des scores) pour chaque groupe. On peut retenir qu'au total, sur la période d'étude, il y a eu 58 participants répartis en 26 participants pour le groupe I (ceux ayant rempli les questionnaires de type 1), 20 participants pour le groupe II (ceux ayant rempli les questionnaires de type 2) et enfin 12 participants pour le groupe III (ceux ayant rempli les questionnaire de type 3). Ce qui est remarquable, c'est que les proportions de questionnaires remplis par type de questionnaires par rapport au nombre total de questionnaires remplis dans une langue donnée sont presque identiques dans les deux langues (anglais et français). On voit aussi que les personnes sollicitées pour répondre aux

questionnaires de type 3 ont moins participé, suivi dans cette faible participation par les personnes sollicitées pour répondre aux questionnaires de type 2, et cela indépendamment de la langue. Cet ordre et ces proportions régulièrement distribués dans les deux langues suggèrent une forme de loi quant à la probabilité d'obtenir une réponse suivant la forme, la présentation d'un questionnaire (y compris le nombre de liens hypertextes présents) car le nombre et le texte des questions posées sont pratiquement identiques pour les différents types de questionnaires. Le tableau 4.3 montre les scores par groupes, calculés en fonction des réponses aux questions Q7, Q8, Q9 et Q10 suivant la méthode présentée au tableau 4.1.

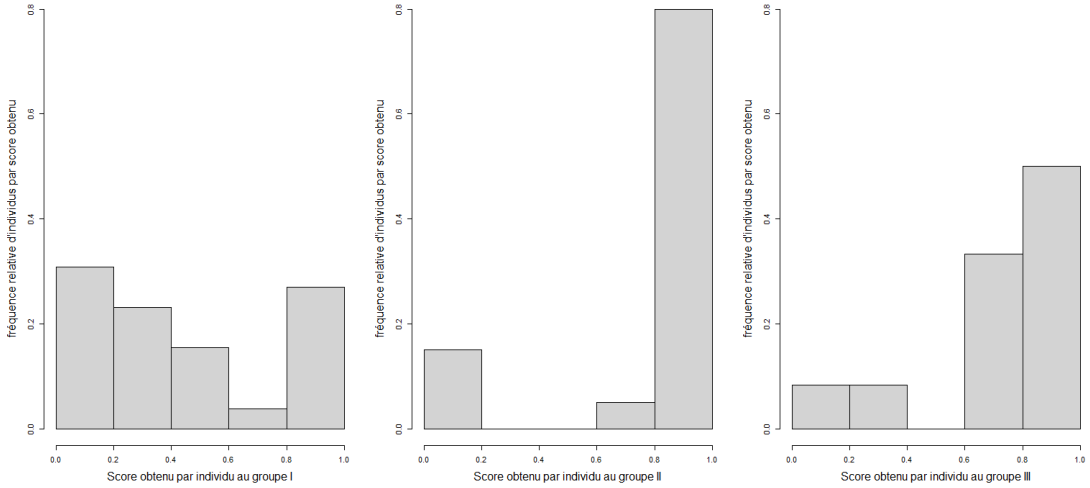


FIGURE 4.2 – Histogramme analyse descriptive des résultats.

4.4 Analyse et résultats

Dans la suite, nous concentrons notre analyse sur les questionnaires de type 1 et les questionnaires de type 2, c'est-à-dire sur le groupe I et le groupe II. Ce choix est suffisant pour trancher la question qui nous préoccupe, à savoir : y a-t-il une différence significative de performance mesurée par les scores, entre ceux

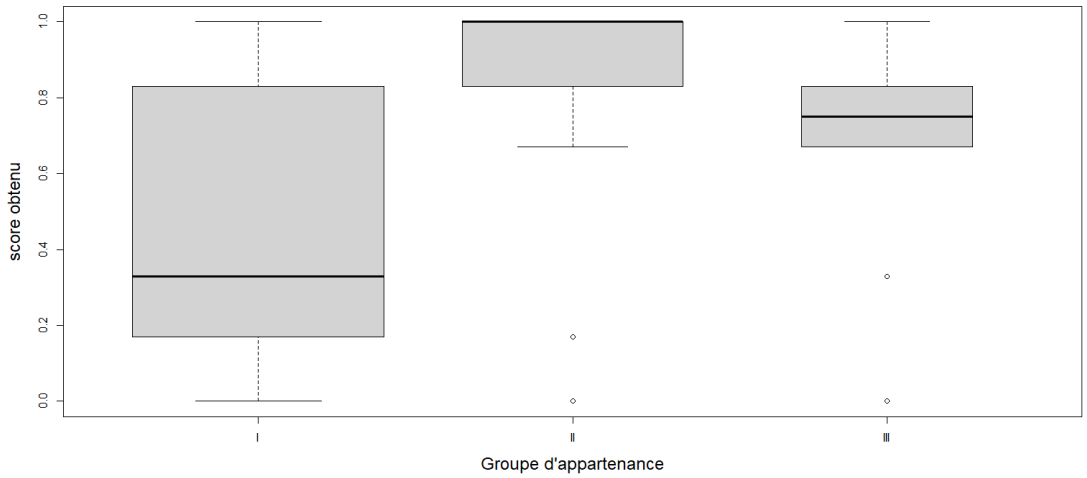


FIGURE 4.3 – Diagramme à moustaches analyse descriptive des résultats.

qui utilisent SIMDGiraffe et ceux qui ne l'utilisent pas ? Le groupe I dans cette étude est le groupe de contrôle et le groupe II est le groupe expérimental. Dans la Figure 4.2, les histogrammes du groupe I et du groupe II et surtout les diagrammes à moustaches (*box-plot* en anglais) de la Figure 4.3 nous montrent très clairement qu'un individu qui utilise SIMDGiraffe a presque toujours un score plus élevé qu'un individu qui n'utilise pas SIMDGiraffe. Nous constatons que les distributions des scores ne sont pas gaussiennes, puisqu'elles ne sont pas symétriques et n'ont pas une forme en cloche. Nous utilisons donc le test de la somme des rangs de Wilcoxon pour deux échantillons indépendants pour comparer les médianes et les moyennes de ces deux groupes (Christensen, 1986, pp. 636-639 ; Olivier, 2018, p. 91). Dans ce test, me_1 désigne la médiane du premier groupe et me_2 désigne la médiane du second groupe, ainsi que sur le tableau 4.2. Les hypothèses sont formulées comme suit :

$$\text{Hypothèse nulle H0} \quad : me_2 = me_1$$

$$\text{Hypothèse alternative H1} \quad : me_2 > me_1$$

Conformément au tableau 4.2, n_1 (respectivement n_2) désignant l'effectif du groupe I (respectivement du groupe II), on a $n_2 < n_1$ et les conditions de la formulation des hypothèses et du test de la somme des rangs de Wilcoxon pour deux échantillons indépendants sont remplies (Christensen, 1986, pp. 636-639). Pour effectuer ce test, nous choisissons un intervalle de confiance de 99 %. Les calculs nous donnent une p-value de $0,54 \times 10^{-3}$, ce qui est inférieur à 0,01. Nous rejetons donc H0, c'est-à-dire que nous acceptons l'hypothèse alternative H1, c'est-à-dire que $me_2 > me_1$ avec 99 % de probabilité. Ce résultat confirme ce que les histogrammes de la Figure 4.2 et les diagrammes à moustaches de la Figure 4.3 montrent déjà. Il confirme également la différence des moyennes entre les deux groupes (Christensen, 1986, pp. 636-639 ; Olivier, 2018, p. 91), et plus exactement dans ce cas précis le fait que $mo_2 > mo_1$ comme le laissent entrevoir ces mêmes figures.

4.5 Discussion et limites

L'utilisation d'une capture d'écran prive le novice de l'interactivité dont il peut disposer avec SIMDGiraffe ; par exemple, il ne peut pas, avec une capture d'écran, choisir de voir le calcul d'un seul champ donné. Cependant, cette utilisation d'une capture d'écran permet de surmonter un problème pratique. La solution de rechange à la capture d'écran nécessiterait que l'expert soit virtuellement ou physiquement présent en même temps que le novice au moment où ce dernier remplit le questionnaire en utilisant SIMDGiraffe. Il serait également nécessaire d'accompagner ces participants dans leur interaction avec SIMDGiraffe. Au-delà du fait que cela est extrêmement compliqué à mettre en œuvre et avec un résultat très incertain en termes de nombre de participants, une étude de validation aléatoire en double aveugle comme celle que nous réalisons devient impossible. Pourtant, nous pensons que par sa rigueur méthodologique et sa robustesse, cette démarche

de validation, comme le suggèrent Seriai *et al.* (2014); Mattila *et al.* (2016); Chotisarn *et al.* (2020) solidifie les bases de la visualisation de logicielle. Cette limitation est atténuée par le fait qu'il s'agit d'une visualisation statique au sens de la taxonomie de Myers (1990) ou de celle de Price *et al.* (1998). Ainsi, l'utilisation d'une capture d'écran ne modifie pas l'information à visualiser. Le novice n'intervient pas dans la génération du contenu des vues, c'est l'interaction de l'expert avec SIMDGiraffe qui génère le contenu de ces vues. Par conséquent, l'utilisation de captures d'écran au lieu de l'interaction avec SIMDGiraffe dans cette étude d'évaluation et de validation avec des novices n'invalide pas, et même n'atténue pas la portée des résultats obtenus.

Nous avons évalué le niveau de connaissance du langage C/C++ des répondants, mais nous n'avons pas utilisé cette évaluation dans nos analyses. Il serait intéressant de voir s'il existe une corrélation entre cette connaissance du langage C/C++ et la compréhension du comportement des *intrinsics*. La réponse à cette question indiquerait s'il est bénéfique ou non d'avoir une certaine connaissance d'un langage tel que le C/C++ avant de chercher à comprendre le fonctionnement des *intrinsics*. Il serait aussi intéressant d'approfondir l'analyse, voire de pousser l'étude plus loin pour comprendre, évaluer et comparer l'apport de la visualisation dans l'acquisition des connaissances d'une part, et dans l'acquisition des compétences d'autre part. Mais tous ces aspects sortent du champ de notre objectif dans le cadre de cette thèse.

TABLEAU 4.3 – Distribution des scores par groupes (types de questionnaires).

Grpe	$S_1^1(Q7)$	$S_1^2(Q8)$	$S_2^1(Q9)$	$S_2^2(Q10)$	S_1	S_2	S
I	1.00	1.00	0.00	0.00	1.00	0.00	0.50
I	1.00	1.00	1.00	1.00	1.00	1.00	1.00
I	1.00	0.00	1.00	0.00	0.33	0.33	0.33
I	1.00	0.00	1.00	0.00	0.33	0.33	0.33
I	1.00	0.00	0.00	0.00	0.33	0.00	0.17
I	1.00	1.00	1.00	1.00	1.00	1.00	1.00
I	1.00	1.00	0.00	1.00	1.00	0.67	0.83
I	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	0.00	0.00	1.00	0.00	0.00	0.33	0.17
I	0.00	0.00	1.00	0.00	0.00	0.33	0.17
I	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	1.00	0.00	0.00	1.00	0.33	0.67	0.50
I	1.00	1.00	1.00	1.00	1.00	1.00	1.00
I	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	1.00	1.00	0.00	1.00	1.00	0.67	0.83
I	0.00	1.00	0.00	0.00	0.67	0.00	0.33
I	1.00	1.00	0.00	0.00	1.00	0.00	0.50
I	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	1.00	1.00	1.00	1.00	1.00	1.00	1.00
I	1.00	1.00	1.00	0.00	1.00	0.33	0.67
I	1.00	1.00	0.00	1.00	1.00	0.67	0.83
I	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	0.00	0.00	0.00	1.00	0.00	0.67	0.33
I	1.00	1.00	0.00	0.00	1.00	0.00	0.50
I	0.00	1.00	0.00	0.00	0.67	0.00	0.33
I	0.00	1.00	0.00	0.00	0.67	0.00	0.33
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	0.00	1.00	1.00	0.67	0.83
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	0.00	1.00	1.00	0.67	0.83
II	0.00	0.00	1.00	0.00	0.00	0.33	0.17
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	0.00	0.00	0.00	0.00	0.00	0.00	0.00
II	1.00	1.00	0.00	1.00	1.00	0.67	0.83
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
II	0.00	0.00	0.00	0.00	0.00	0.00	0.00
II	1.00	1.00	1.00	0.00	1.00	0.33	0.67
II	1.00	1.00	1.00	1.00	1.00	1.00	1.00
III	1.00	1.00	0.00	1.00	1.00	0.67	0.83
III	0.00	1.00	0.00	1.00	0.67	0.67	0.67
III	1.00	1.00	1.00	1.00	1.00	1.00	1.00
III	0.00	1.00	0.00	1.00	0.67	0.67	0.67
III	0.00	1.00	0.00	1.00	0.67	0.67	0.67
III	1.00	1.00	1.00	1.00	1.00	1.00	1.00
III	0.00	0.00	0.00	0.00	0.00	0.00	0.00
III	0.00	1.00	0.00	0.00	0.67	0.00	0.33
III	1.00	1.00	0.00	1.00	1.00	0.67	0.83
III	0.00	1.00	0.00	1.00	0.67	0.67	0.67
III	1.00	1.00	0.00	1.00	1.00	0.67	0.83
III	1.00	1.00	0.00	1.00	1.00	0.67	0.83

CONCLUSION

Le potentiel offert par la programmation vectorielle (architecture SIMD) semble omniprésent ; des ordinateurs aux téléphones intelligents. Mais actualiser ce potentiel en écrivant des programmes qui l'exploite de manière optimale reste une gageure en raison du caractère rebutant de ce paradigme de programmation. Ce caractère rebutant de la programmation SIMD se manifeste à la fois par la difficulté à comprendre le fonctionnement des *intrinsics* et le comportement de codes produits avec ces *intrinsics*. Nous avons essayé d'apporter notre contribution dans l'aplanissement de cette double difficulté en l'abordant du point de vue de l'expert et du novice en programmation vectorielle, et en mobilisant pour cela les outils de la visualisation de logiciel. Mais ce faisant nous avons aussi saisi cette opportunité pour renforcer la « ceinture protectrice » (pour emprunter l'expression de Lakatos (1980)) de la SV, et par extension de l'InfoViz, en élargissant sa pratique et en corroborant ses hypothèses sur un observable. Nous avons de cette façon contribué au renforcement des fondements de la SV (et par ricochet de l'InfoViz) en tant que science.

Ainsi, en ce qui a trait à la compréhension du fonctionnement des fonctions vectorielles ou *intrinsics*, en nous appuyant sur le modèle de ces fonctions que nous avons développées et suivant la méthodologie préconisée par Munzner (2009); Sedlmair *et al.* (2012), nous avons conçu et implémenté le module 2 de SIMDGiraffe. Ce module est un apport dans l'aide à la compréhension du fonctionnement des *intrinsics* par un novice. La conception et l'implémentation de ce module constituent aussi une contribution à la visualisation de logiciel du fait de son originalité qui consiste à utiliser un outil de visualisation pour capturer la pensée

d'un expert. En effet, quoiqu'affirmée très tôt comme étant un des objectifs de la visualisation de logiciel (Petre et Blackwell, 1998), cette possibilité n'avait pas été jusqu'ici explorée à notre connaissance, y compris en InfoViz. Dans cette optique, l'expert devient un acteur dans l'utilisation de l'outil de visualisation dans son rôle d'expert. Ceci est à distinguer des situations traditionnelles comme dans (Hakone *et al.*, 2016) où l'expert du domaine utilise l'outil de visualisation, mais dans le rôle de n'importe quel utilisateur. En fait dans le module 2 de SIMDGiraffe, la connaissance à mettre à la disposition du novice à travers cet outil est dans la tête de l'expert ; d'où la nécessité de l'interaction expert/SIMDGiraffe comme préalable à l'utilisation de l'outil par le novice.

Ensuite, en ce qui concerne la compréhension du comportement d'un code écrit avec les *intrinsics*, partant là aussi d'un modèle de comportement d'un code s'exécutant sur une architecture donnée que nous avons développé, nous avons conçu et implémenté le module 1 du prototype SIMDGiraffe. Nous avons validé ce module 1 par deux scénarios de cas d'utilisation, conformément à la pratique communément acceptée en la matière (Sedlmair *et al.*, 2012; Munzner, 2009). Il convient de souligner qu'outre les modules mentionnés, tous disponibles en ligne sur GitHub (`https://pmntang.github.io/SIMDGiraffe` pour les versions live et `https://github.com/pmntang/SIMDGiraffe` pour les versions de test), les modèles développés lors de la mise en œuvre de ces modules sont en soit des contributions. Nous avons montré pour le modèle relatif aux fonctions vectorielles comment il peut être traduit en une ontologie en vue de faire des raisonnements sur ces fonctions ou même en vue d'être utilisé dans des applications intelligentes. L'importance de cette contribution scientifique des modèles au domaine de l'InfoViz (et donc de la SV) est bien saisie par Munzner (2008) qui recommande ainsi aux organisateurs des conférences et aux éditeurs de journaux scientifiques de la visualisation d'information d'accorder une plus grande place aux articles portant sur

ce type de contribution. Nos travaux autour du module 1 de SIMDGiraffe et du modèle y afférent ont fait l'objet d'une présentation à la conférence Information Visualization Theory and Applications (IVAPP) 2021 et d'une publication scientifique dont le DOI est : 10.5220/0010195201470154. Pour ce qui est de la SV en tant que science, elle a pris un risque de falsification à travers l'expérimentation menée auprès de novices dans la compréhension du fonctionnement des *intrinsics* avec l'aide d'un outil développé selon les prescriptions de la SV d'une part et sans l'aide de cet outil d'autre part. La corroboration des hypothèses de la SV obtenue à la suite de cette expérimentation renforce à la fois les fondements de cette science et élargit son domaine de validité, ce qui est une contribution importante.

Plusieurs interrogations ont jailli au cours de notre démarche. Par exemple, nous aurions voulu comprendre l'interaction que peuvent avoir l'explication d'un expert et la visualisation, ce qu'aurait dû nous permettre le questionnaire de type 3. Mais nous n'avions pas pris en compte un facteur important que nous n'avons pas jusqu'ici pu expliquer, mais que nous avons observé chez les individus ayant rempli ce questionnaire. En effet, alors que les questions sont identiques pour les trois types de questionnaires, on observe un comportement très particulier des individus du groupe 3 (c'est-à-dire ceux ayant rempli le questionnaire de type 3). Le nombre de personnes de ce groupe qui remplissent et soumettent leur questionnaire est très faible, peu importe la langue. D'ailleurs en proportion du nombre total de personnes ayant répondu, ce nombre est similaire si l'on partage les questionnaires suivant les deux langues. De façon générale, les proportions par groupe de personnes ayant effectivement répondu sont similaires dans les deux langues. Cela laisse suggérer que la présentation du questionnaire (y compris la présence ou non de liens hypertextes) a une influence sur le participant quant à sa décision de soumettre ou non son questionnaire, c'est-à-dire de répondre [4]. En tout

4. Nous faisons une différence entre participer et soumettre le questionnaire. En effet, nous

cas, sans une claire élucidation des facteurs qui déterminent ce comportement, il nous a semblé peu pertinent d'aller de l'avant avec l'analyse de l'influence de l'explication d'un expert sur la visualisation pour un novice.

Nous pensons à terme faire évoluer SIMDGiraffe du stade de prototype vers un outil mature et en assurer la promotion. Cela ne veut pas dire qu'il n'est pas utile ou utilisable actuellement, mais il s'agit de l'améliorer et par exemple de l'intégrer dans un environnement comme Godbolt (`https://godbolt.org/`) ou même dans le site web d'un fabricant comme Intel®.

Enfin, la science a progressé pendant nos travaux avec l'avènement ou la généralisation des modèles d'IA capables de générer du texte et même des images pour répondre à une requête : on peut mentionner le nouveau Bing de Microsoft (`https://www.bing.com/new`), ChatGPT (`https://openai.com/chatgpt`), etc. Il serait intéressant par exemple d'entrainer un modèle capable de générer des images pour expliquer le comportement de code, notamment de codes vectoriels, à la demande.

---

avons estimé la participation par le nombre de vues des vidéos explicatives sur YouTube qui est passé de 5 vues à 365 vues pendant la période de l'étude. Ce nombre de vues n'avait pas beaucoup bougé avant cette période, et il n'a plus beaucoup bougé après cette période. Bien entendu cette estimation ne concerne que les questionnaires de type 2 et les questionnaires de type 3 qui comportent des liens vers ces vidéos.

APPENDICE A

CERTIFICAT D'ÉTHIQUE

CERTIFICAT D'ÉTHIQUE

2022-08

Le comité d'éthique de la recherche de la Téluq certifie avoir examiné la proposition de recherche soumise par

Pierre Marie Ntang

Intitulée

SIMDGiraffe :Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions

et avoir conclu que la recherche proposée est entièrement conforme aux normes d'éthique en recherche selon la *Politique d'éthique de la recherche avec les êtres humains*.

Valide jusqu'au 12 avril 2023

2022-04-12

Date

François Pichette
Président

APPENDICE B

LES QUESTIONNAIRES UTILISÉS

QuestionnaireGroup1

Thank you for taking part in this study which aims to understand the influence that visualization can have on the understanding of software behavior. You are kindly requested to answer 12 questions whose answers range from automatic completions to some calculations that you can do mentally. You can also use a calculator or an Excel spreadsheet. If you wish, you can answer this questionnaire anonymously by providing a pseudonym instead of your name. This study has received ethical certification from the Ethics Committee for Research with Human Beings of TELUQ University (CER-TELUQ) number 2022-08 of April 12, 2022.

A- Personal information, start date and start time of filling out the questionnaire

Before moving on to the next questions, these questions must be filled in first. Questions Q2 and Q3 are automatically filled in with the current date and current time respectively; hence, you only have to fill in Q1.

Q1- Name or pseudonym *

Name or pseudonym

Enter your name in the space above, or a pseudonym if you want to remain anonymous.

Q2-Date : *

2022-07-25

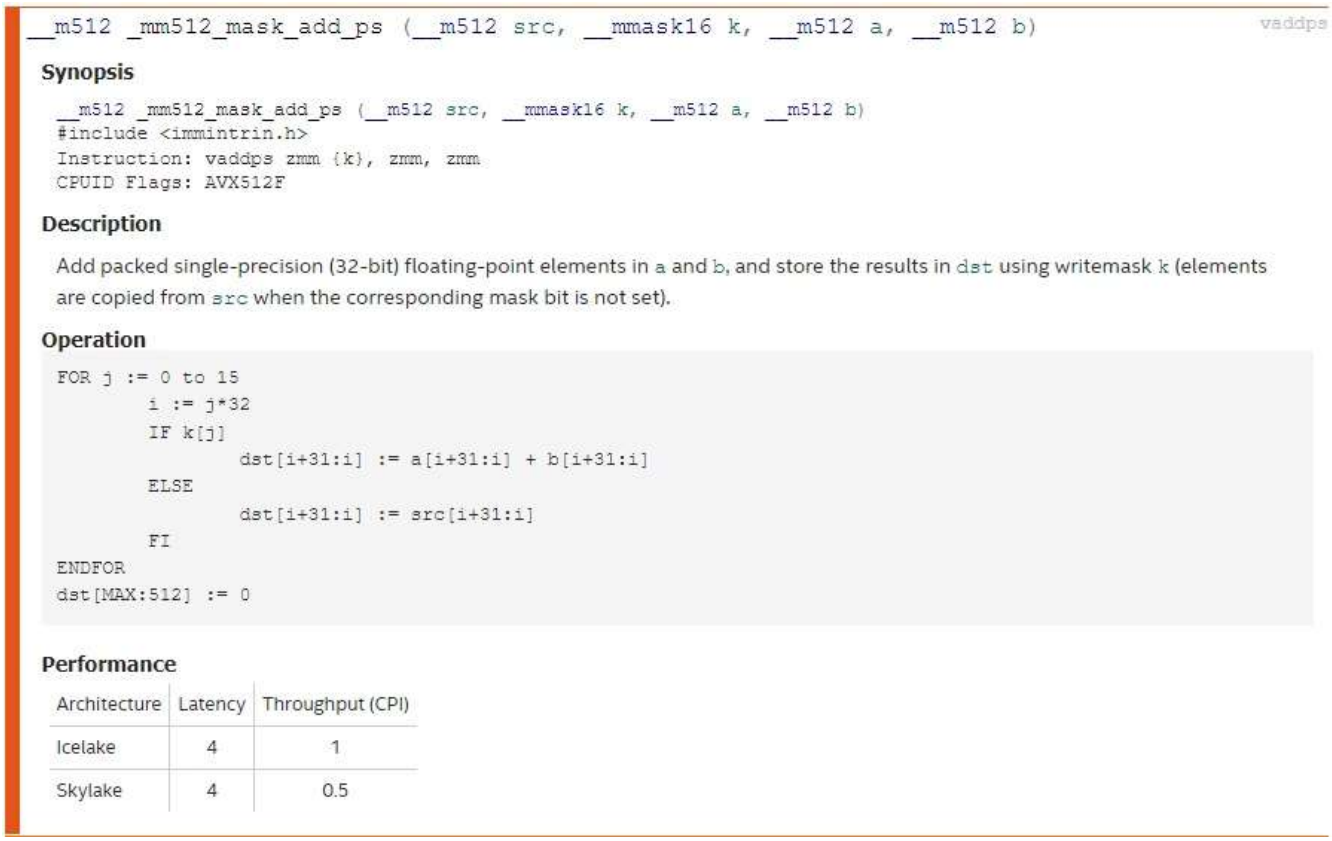
Q3-Start time: *

3:29 PM

C- Understanding the behavior of the vector instructions _mm512_mask_add_ps and _mm_shuffle_epi32

1- Vector instruction _mm512_mask_add_ps

Carefully read the explanation of this instruction on the figure below. You can also find this explanation on Intel © web site. Make sure you read and understand the explanation before answering questions Q7 and Q8. You can also watch this short video where the instruction is explained; the video is an explanation made by an expert in the field of vector programming. You should only use the explanations provided (you can of course consult the Intel © web site), but do not use other resources (for example search on Google, other documents, etc.).



__m512 _mm512_mask_add_ps (__m512 src, __mmask16 k, __m512 a, __m512 b) vaddps

Synopsis

```
__m512 _mm512_mask_add_ps (__m512 src, __mmask16 k, __m512 a, __m512 b)
#include <immintrin.h>
Instruction: vaddps zmm {k}, zmm, zmm
CPUID Flags: AVX512F
```

Description

Add packed single-precision (32-bit) floating-point elements in a and b, and store the results in dst using writemask k (elements are copied from src when the corresponding mask bit is not set).

Operation

```
FOR j := 0 to 15
        i := j*32
        IF k[j]
                dst[i+31:i] := a[i+31:i] + b[i+31:i]
        ELSE
                dst[i+31:i] := src[i+31:i]
        FI
ENDFOR
dst[MAX:512] := 0
```

Performance

Architecture	Latency	Throughput (CPI)
Icelake	4	1
Skylake	4	0.5

Q7- After reading the description and the explanation above, say what the _mm512_mask_add_ps instruction does by performing the following calculation: given src=(1, 3, 4, 1, 2, 5, 4, 1, 2, 3 , 4, 1, 1, 3, 4, 1); k=(1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0); a=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1); b=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) . Calculate r = _ mm512_mask_add_ps(src, k, a, b) *

◯ r = (1, 0, 0, 1, 2, 0, 0, 1, 0, 3, 4, 0, 0, 0, 4, 0) ◯ r = (12, 2, 4, 6, 2, 8, 10, 2, 4, 6, 8, 2, 6, 2, 4, 0)

◯ r = (12, 3, 4, 6, 2, 5, 4, 2, 2, 6, 8, 1, 1, 3, 4, 1) ◯ r = (13, 0, 0, 7, 4, 0, 0, 3, 0, 9, 12, 0, 0, 0, 8, 0)

Check the radio button in front of the correct answer.

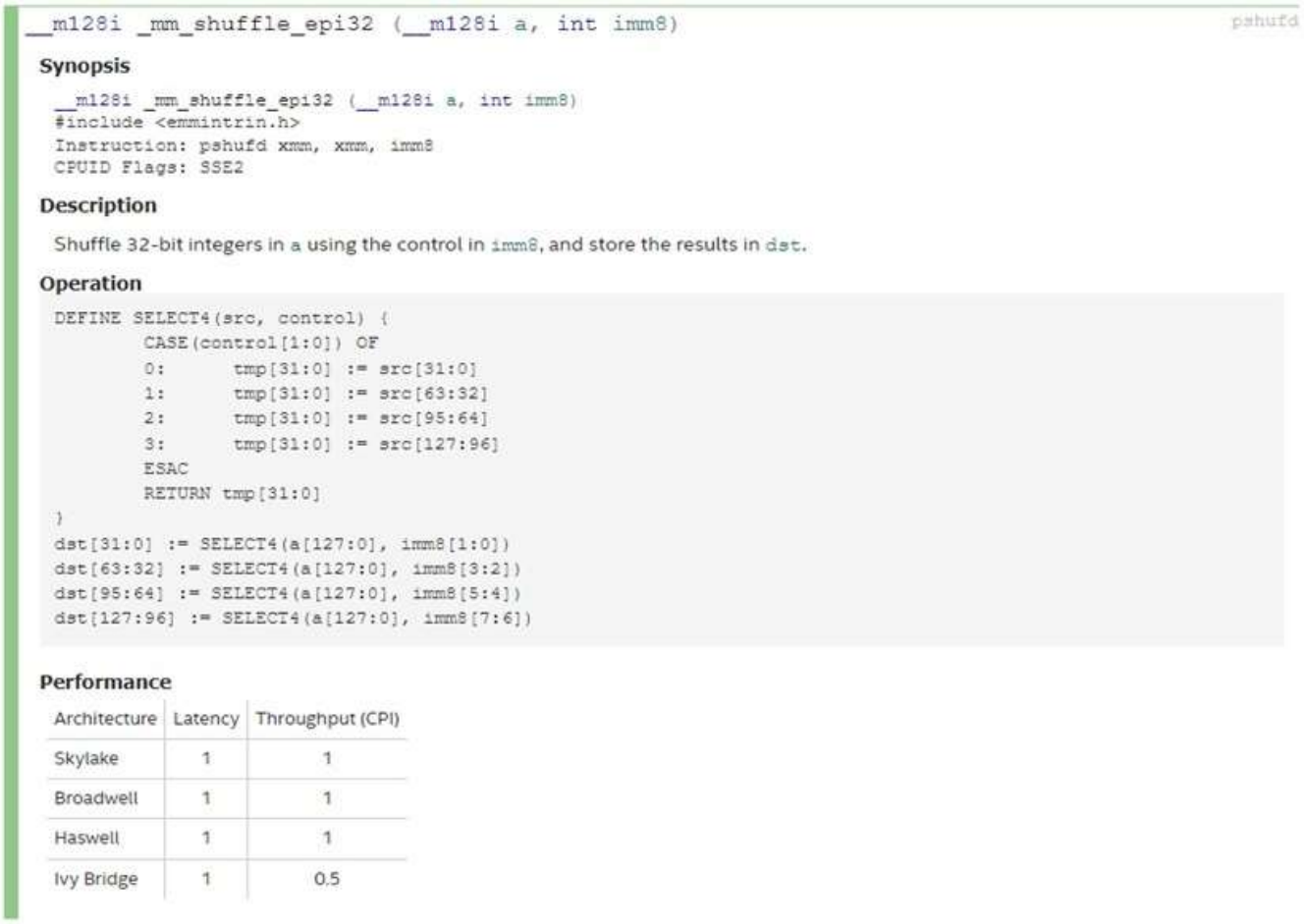
Q8- Using the description and the explanation above again, give a general formula for calculating the coordinates of r(ri) as function of those of src(srci), k(ki), a(ai) and b(bi)). ri=? *

◯ ri=ai+bi ◯ ri=(1-ki) x srci+ai+bi ◯ ri=(1-ki) x srci+ki x (ai+bi) ◯ ri=ki x srci+(1-ki) x (ai+bi)

Check the radio button in front of the correct answer.

2-Vector instruction _mm_shuffle_epi32

Carefully read the explanation of this instruction on the figure below. You can find this explanation on Intel © web site. Make sure you read and understand the explanation before answering questions Q9 and Q10. You can also watch this short video where the instruction is explained; the video is an explanation made by an expert in the field of vector programming. You should only use the explanations provided (you can naturally consult the Intel © site by following the link given above), but do not use other resources (for example search on Google, other documents, etc.).



__m128i _mm_shuffle_epi32 (__m128i a, int imm8) pshufd

Synopsis

```
__m128i _mm_shuffle_epi32 (__m128i a, int imm8)
#include <emmintrin.h>
Instruction: pshufd xmm, xmm, imm8
CPUID Flags: SSE2
```

Description

Shuffle 32-bit integers in a using the control in imm8, and store the results in dst.

Operation

```
DEFINE SELECT4(src, control) {
        CASE(control[1:0]) OF
        0:      tmp[31:0] := src[31:0]
        1:      tmp[31:0] := src[63:32]
        2:      tmp[31:0] := src[95:64]
        3:      tmp[31:0] := src[127:96]
        ESAC
        RETURN tmp[31:0]
}
dst[31:0] := SELECT4(a[127:0], imm8[1:0])
dst[63:32] := SELECT4(a[127:0], imm8[3:2])
dst[95:64] := SELECT4(a[127:0], imm8[5:4])
dst[127:96] := SELECT4(a[127:0], imm8[7:6])
```

Performance

Architecture	Latency	Throughput (CPI)
Skylake	1	1
Broadwell	1	1
Haswell	1	1
Ivy Bridge	1	0.5

Q9- After reading the description and the explanation above, say what the _mm_shuffle_epi32 instruction does by performing the following calculation: given a=(6, 7, 4, 3); imm8=(0, 1, 2, 3) . Calculate r = _mm_shuffle_epi32(a, imm8) *

◯ r = (6, 2, 1, 3) ◯ r = (6, 7, 4, 3) ◯ r = (3, 4, 7, 6) ◯ r = (3, 7, 4, 6)

Check the radio button in front of the correct answer.

Q10- Usingthe description and the explanation above again, give a general formula for calculating the coordinates of r(ri) as function of those of a(ai) and imm8(imm8i). ri=? *

◯ ri=aij, where j=imm8i ◯ ri=ai ◯ ri=ai x imm8i ◯ ri=aj, where j=imm8i

Check the radio button in front of the correct answer

B- Preliminary knowledge

I- Knowledge of algebra and vector space

Consider the real vector space R3. For A, B, C, Res1, Res2, five vectors of R3 such that A=(a1, a2, a3), B=(b1, b2, b3), C=(c1, c2, c3), Res1=(x1, x2, x3), Res2=(y1, y2, y3) we define vectSum(A,B,C)=Res1 and vectProd(A,B,C)=Res2 by

$$\begin{cases} x_1 = a_1 - b_1 + c_1 \\ x_2 = a_2 - b_2 + c_2 \\ x_3 = a_3 - b_3 + c_3 \end{cases} \text{ and } \begin{cases} y_1 = b_1 \times (a_1 - c_1) + c_1 \\ y_2 = b_2 \times (a_2 - c_2) + c_2 \\ y_3 = b_3 \times (a_3 - c_3) + c_3 \end{cases}$$

Now let's assume that A=(1, 0,1) ; B=(1,1,0) ; C=(0,1, 1).

Q4- Calculate each of the Res1 and Res2 vectors: Res1= ? Res2=? *

- ◯ Res1=(2,1,0); Res2=(1,1,0).
- ◯ Res1=(0,0,2); Res2=(1,0,1).
- ◯ Res1=(2,2,2); Res2=(1,1,1).
- ◯ Res1=(1,0,2); Res2=(0,0,1).

Check the radio button in front of the correct answer

Q5- Give a general formula for calculating the coordinates of Res1(xi) and Res2(yi) as a function of those of A (ai), B (bi) and C (ci). xi=? yi=? *

xi= ; yi= .

Write xi= ; yi= . Then, Write the expression of xi (respectively yi) as function of ai, bi and ci in the space following xi (respectively yi).

II- Knowledge of the C language

Consider the following function f in C: int f (int x, int y) {return x-y;}.

Q6- Choose the two instructions in C (that is, instruction1 and instruction2) which allow you to declare three integer variables a, b, c and to place in c the difference between a and b using the function f. instruction1: ? instruction2: ? *

- ◯ Instruction1: int c, a, b; Instruction2: c=f(a-b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a,b);}
- ◯ Instruction1: int c, a, b; Instruction2: c=f(a,b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a-b);}

Check the radio button in front of the correct answer

D- End time of the questionnaire completion and comments

Before submitting the forms, these questions, in particular questions *Q18* must be filled in.

Q11- End time: *

End time

Fill in this field with the current time, to the nearest minute, when you have finished filling out the questionnaire and if you have completed it without interruption. If you have had interruptions, calculate the end time by deducting the duration of the total interruptions from the current time.

Q12- Other comments and remarks:

Fill in this field with your remarks, comments, and observations on any subject of interest in connection with the study, including the questionnaire, the SIMDGiraffe prototype, etc.

Submit

QuestionnaireGroup2

Thank you for taking part in this study which aims to understand the influence that visualization can have on the understanding of software behavior. You are kindly requested to answer 12 questions whose answers range from automatic completions to some calculations that you can do mentally. You can also use a calculator or an Excel spreadsheet. If you wish, you can answer this questionnaire anonymously by providing a pseudonym instead of your name. This study has received ethical certification from the Ethics Committee for Research with Human Beings of TELUQ University (CER-TELUQ) number 2022-08 of April 12, 2022.

A- Personal information, start date and start time of filling out the questionnaire

Before moving on to the next questions, these questions must be filled in first. Questions Q2 and Q3 are automatically filled in with the current date and current time respectively; hence, you only have to fill in Q1.

Q1- Name or pseudonym *

Name or pseudonym

Enter your name in the space above, or a pseudonym if you want to remain anonymous.

Q2-Date : *

2022-07-25

Q3-Start time: *

3:36 PM

C- Understanding the behavior of the vector instructions _mm512_mask_add_ps and _mm_shuffle_epi32

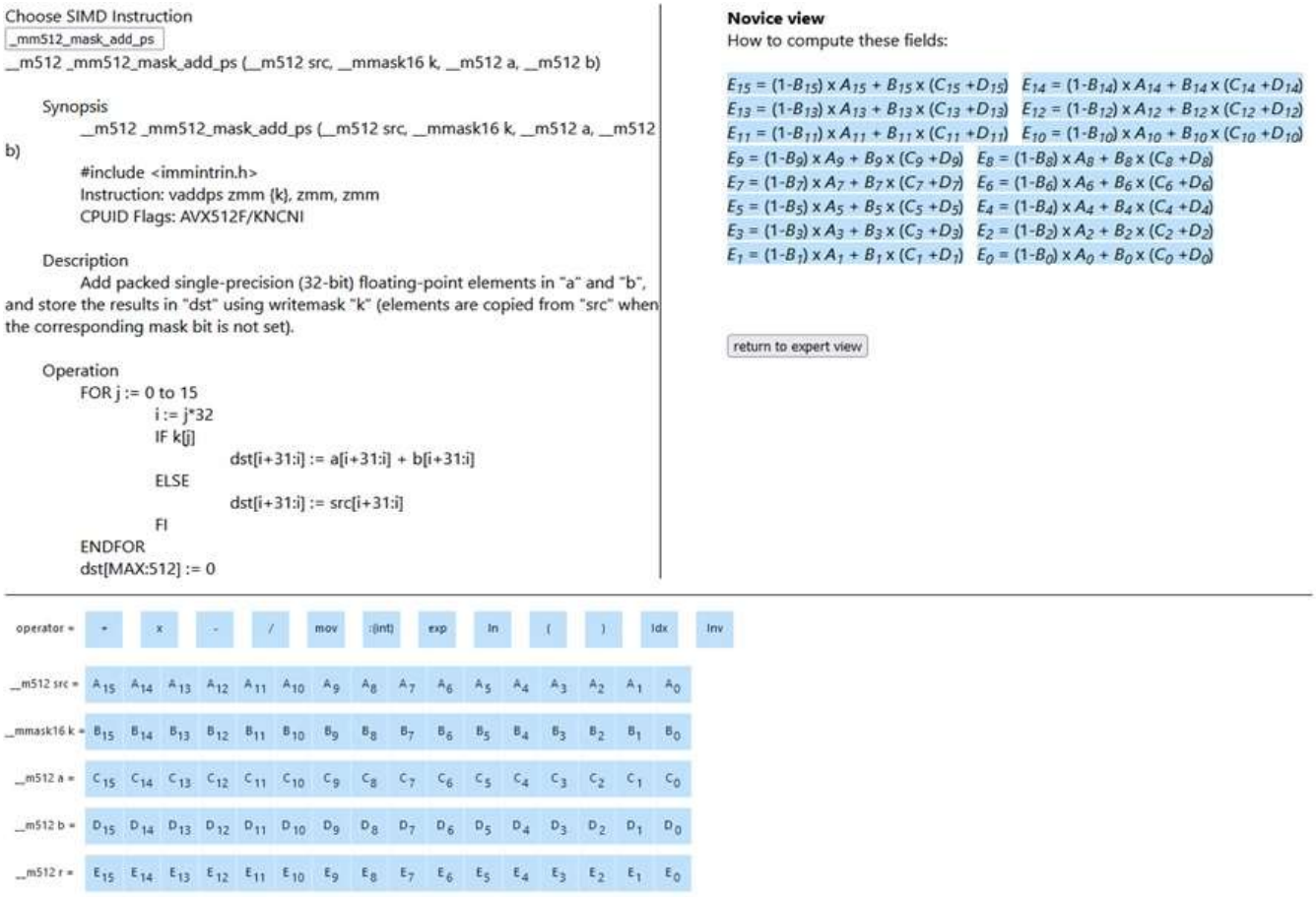
1- Vector instruction _mm512_mask_add_ps

Before answering questions Q7 and Q8, carefully observe and try to understand the figure below which is a screenshot of the SIMDGiraffe prototype, available online at https://github.com/pmntang/SIMDGiraffe. This screenshot is divided into three dials.

On the left dial there is a description of the instruction provided by Intel©.

On the lower dial, there is the graphical translation of this description. This translation consists in displaying for each of these vectors, its fields (or coordinates). We use the letters of the alphabet with subscripts (A0, A1, …B0, B1,…, ...) written inside blue rectangles to designate these fields. The graphic description is preceded on the line, to the left of the equality sign (=), by the name of the vector in question (src, k, a, b, r) and its type (__m512, __mmask16, __m512, __m512, __m512).

On the right-hand dial there is a visual description of the links between each field (or coordinates) of the result vector r and the fields (or coordinates) of the operand vectors used to calculate this field (or this coordinate). This description consists, as we can see, in giving for each field of the vector result r the calculation formula of that field from the operand fields used to carry out this calculation.



Q7- After observing the figure above, say what the _mm512_mask_add_ps instruction does by performing the following calculation: given src=(1, 3, 4, 1, 2, 5, 4, 1, 2, 3 , 4, 1, 1, 3, 4, 1); k=(1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0); a=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1); b=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) . Calculate r = _mm512_mask_add_ps(src, k, a, b) *

- ◯ r = (1, 0, 0, 1, 2, 0, 0, 1, 0, 3, 4, 0, 0, 0, 4, 0)
- ◯ r = (12, 2, 4, 6, 2, 8, 10, 2, 4, 6, 8, 2, 6, 2, 4, 0)
- ◯ r = (12, 3, 4, 6, 2, 5, 4, 2, 2, 6, 8, 1, 1, 3, 4, 1)
- ◯ r = (13, 0, 0, 7, 4, 0, 0, 3, 0, 9, 12, 0, 0, 0, 8, 0)

Check the radio button in front of the correct answer. You can use the right quadrant of the previous figure as a help.

Q8- Using the figure above again, give a general formula for calculating the coordinates of r(ri) as function of those of src(srci), k(ki), a(ai) and b(bi)). ri=? *

- ◯ ri=Ei=Ci+Di=ai+bi
- ◯ ri=Ei=(1-Bi) x Ai+Ci+Di=(1-ki) x srci+ai+bi
- ◯ ri=Ei=(1-Bi) x Ai+Bi x (Ci+Di)=(1-ki) x srci+ki x (ai+bi)
- ◯ ri=Ei=Bi x Ai+(1-Bi) x (Ci+Di)=ki x srci+(1-ki) x (ai+bi)

Check the radio button in front of the correct answer; pay attention to the fact that on the previous figure r=(Ei)i=r(ri); b=(Di)i=b(bi); a=(Ci)i=a(ai); k=(Bi)i=k(ki); src=(Ai)i=src(srci). i being the index.

2-Vector instruction _mm_shuffle_epi32

Before answering questions *Q9* and *Q10*, carefully observe and try to understand the figure below which is a screenshot of the SIMDGiraffe prototype, available online at https://github.com/pmntang/SIMDGiraffe. This screenshot is divided into three dials.

On the left dial there is a description of the instruction provided by Intel©.

On the lower dial, there is the graphical translation of this description. This translation consists in displaying for each of these vectors, its fields (or coordinates). We use the letters of the alphabet with subscripts (A0, A1, …B0, B1,…, ...) written inside blue rectangles to designate these fields. The description graphic is preceded on the line, to the left of the equality sign (=), by the name of the vector in question (a, imm8, r) and its type (__m128i, int, __m128i).

On the right-hand dial there is a visual description of the links between each field (or coordinate) of the result vector r and the fields (or coordinates) of the operand vectors used to calculate this field (or this coordinate). This description consists, as we can see, in giving for each field of the vector result r the calculation formula of that field from the operand fields used to carry out this calculation.

Q9- After observing the figure above, say what the _mm_shuffle_epi32 instruction does by performing the following calculation: given a=(6, 7, 4, 3); imm8=(0, 1, 2, 3) . Calculate r = _mm_shuffle_epi32(a, imm8) *

◯ r = (6, 2, 1, 3) ◯ r = (6, 7, 4, 3) ◯ r = (3, 4, 7, 6) ◯ r = (3, 7, 4, 6)

Check the radio button in front of the correct answer. You can use the right quadrant of the previous figure as a help.

Q10- Using the figure above again, give a general formula for calculating the coordinates of r(ri) as function of those of a(ai) and imm8(imm8i). ri=? *

- ri=Ci=Aij=aij, where j=Bi=imm8i
- ri=Ci=Ai=ai
- ri=Ci=Ai x Bi=ai x imm8i
- ri=Ci=Aj=aj, where j=Bi=imm8i

Check the radio button in front of the correct answer; pay attention to the fact that on the previous figure r=(Ci)i=r(ri); a=(Ai)i=a(ai); imm8=(Bi)i=imm8(imm8i). i being the index.

B- Preliminary knowledge

I- Knowledge of algebra and vector space

Consider the real vector space R3. For A, B, C, Res1, Res2, five vectors of R3 such that A=(a1, a2, a3), B=(b1, b2, b3), C=(c1, c2, c3), Res1=(x1, x2, x3), Res2=(y1, y2, y3) we define vectSum(A,B,C)=Res1 and vectProd(A,B,C)=Res2 by

$$\begin{cases} x_1 = a_1 - b_1 + c_1 \\ x_2 = a_2 - b_2 + c_2 \\ x_3 = a_3 - b_3 + c_3 \end{cases} \text{ and } \begin{cases} y_1 = b_1 \times (a_1 - c_1) + c_1 \\ y_2 = b_2 \times (a_2 - c_2) + c_2 \\ y_3 = b_3 \times (a_3 - c_3) + c_3 \end{cases}$$

Now let's assume that A=(1, 0,1) ; B=(1,1,0) ; C=(0,1, 1).

Q4- Calculate each of the Res1 and Res2 vectors: Res1= ? Res2=? *

- Res1=(2,1,0); Res2=(1,1,0).
- Res1=(0,0,2); Res2=(1,0,1).
- Res1=(2,2,2); Res2=(1,1,1).
- Res1=(1,0,2); Res2=(0,0,1).

Check the radio button in front of the correct answer

Q5- Give a general formula for calculating the coordinates of Res1(xi) and Res2(yi) as a function of those of A (ai), B (bi) and C (ci). xi=? yi=? *

xi= ; yi= .

Write xi= ; yi= . Then, Write the expression of xi (respectively yi) as function of ai, bi and ci in the space following xi (respectively yi).

II- Knowledge of the C language

Consider the following function f in C: int f (int x, int y) {return x-y;}.

Q6- Choose the two instructions in C (that is, instruction1 and instruction2) which allow you to declare three integer variables a, b, c and to place in c the difference between a and b using the function f. instruction1: ? instruction2: ? *

- ◯ Instruction1: int c, a, b; Instruction2: c=f(a-b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a,b);}
- ◯ Instruction1: int c, a, b; Instruction2: c=f(a,b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a-b);}

Check the radio button in front of the correct answer

D- End time of the questionnaire completion and comments

Before submitting the forms, these questions, in particular questions *Q18* must be filled in.

Q11- End time: *

End time

Fill in this field with the current time, to the nearest minute, when you have finished filling out the questionnaire and if you have completed it without interruption. If you have had interruptions, calculate the end time by deducting the duration of the total interruptions from the current time.

Q12- Other comments and remarks:

Fill in this field with your remarks, comments, and observations on any subject of interest in connection with the study, including the questionnaire, the SIMDGiraffe prototype, etc.

Submit

QuestionnaireGroup3

Thank you for taking part in this study which aims to understand the influence that visualization can have on the understanding of software behavior. You are kindly requested to answer 12 questions whose answers range from automatic completions to some calculations that you can do mentally. You can also use a calculator or an Excel spreadsheet. If you wish, you can answer this questionnaire anonymously by providing a pseudonym instead of your name. This study has received ethical certification from the Ethics Committee for Research with Human Beings of TELUQ University (CER-TELUQ) number 2022-08 of April 12, 2022.

A- Personal information, start date and start time of filling out the questionnaire

Before moving on to the next questions, these questions must be filled in first. Questions Q2 and Q3 are automatically filled in with the current date and current time respectively; hence, you only have to fill in Q1.

Q1- Name or pseudonym *

Name or pseudonym

Enter your name in the space above, or a pseudonym if you want to remain anonymous.

Q2-Date : *

2022-07-25

Q3-Start time: *

3:37 PM

C- Understanding the behavior of the vector instructions _mm512_mask_add_ps and _mm_shuffle_epi32

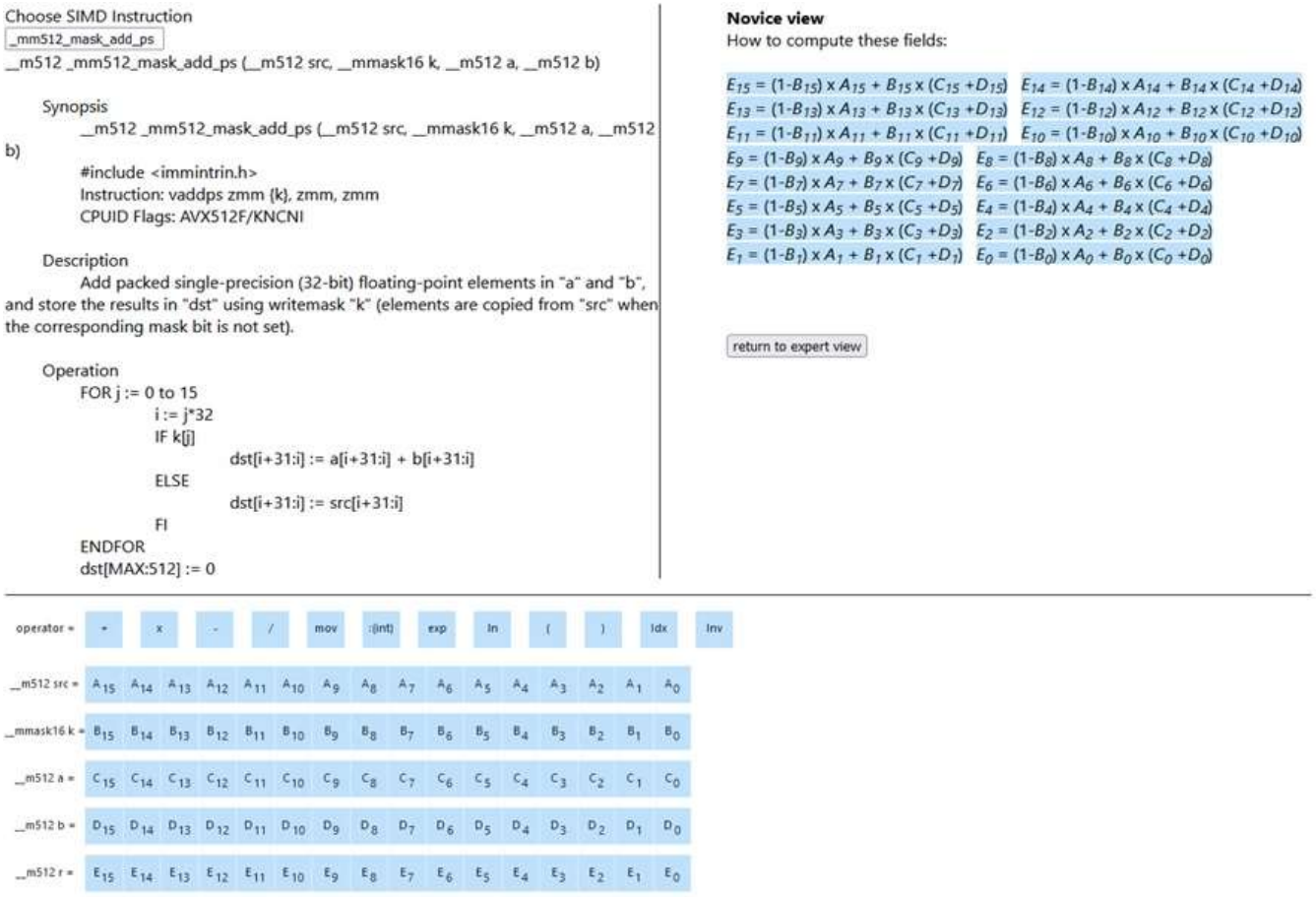
1- Vector instruction _mm512_mask_add_ps

Before answering questions Q7 and Q8, carefully observe and try to understand the figure below which is a screenshot of the SIMDGiraffe prototype, available online at https://github.com/pmntang/SIMDGiraffe. This screenshot is divided into three dials.

On the left dial there is a description of the instruction provided by Intel©.

On the lower dial, there is the graphical translation of this description. This translation consists in displaying for each of these vectors, its fields (or coordinates). We use the letters of the alphabet with subscripts (A0, A1, …B0, B1,…, ...) written inside blue rectangles to designate these fields. The graphic description is preceded on the line, to the left of the equality sign (=), by the name of the vector in question (src, k, a, b, r) and its type (__m512, __mmask16, __m512, __m512, __m512).

On the right-hand dial there is a visual description of the links between each field (or coordinates) of the result vector r and the fields (or coordinates) of the operand vectors used to calculate this field (or this coordinate). This description consists, as we can see, in giving for each field of the vector result r the calculation formula of that field from the operand fields used to carry out this calculation. You can also watch this short video which provides and explanation by an expert in the field of vector programming of this instruction: https://youtu.be/aoAX922SeGs



Q7- After observing the figure above, say what the _mm512_mask_add_ps instruction does by performing the following calculation: given src=(1, 3, 4, 1, 2, 5, 4, 1, 2, 3 , 4, 1, 1, 3, 4, 1); k=(1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0); a=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1); b=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) . Calculate r = _mm512_mask_add_ps(src, k, a, b) *

- ◯ r = (1, 0, 0, 1, 2, 0, 0, 1, 0, 3, 4, 0, 0, 0, 4, 0)
- ◯ r = (12, 2, 4, 6, 2, 8, 10, 2, 4, 6, 8, 2, 6, 2, 4, 0)
- ◯ r = (12, 3, 4, 6, 2, 5, 4, 2, 2, 6, 8, 1, 1, 3, 4, 1)
- ◯ r = (13, 0, 0, 7, 4, 0, 0, 3, 0, 9, 12, 0, 0, 0, 8, 0)

Check the radio button in front of the correct answer. You can use the right quadrant of the previous figure as a help.

Q8- Using the figure above again, give a general formula for calculating the coordinates of r(ri) as function of those of src(srci), k(ki), a(ai) and b(bi)). ri=? *

- ◯ ri=Ei=Ci+Di=ai+bi
- ◯ ri=Ei=(1-Bi) x Ai+Ci+Di=(1-ki) x srci+ai+bi
- ◯ ri=Ei=(1-Bi) x Ai+Bi x (Ci+Di)=(1-ki) x srci+ki x (ai+bi)
- ◯ ri=Ei=Bi x Ai+(1-Bi) x (Ci+Di)=ki x srci+(1-ki) x (ai+bi)

Check the radio button in front of the correct answer; pay attention to the fact that on the previous figure r=(Ei)i=r(ri); b=(Di)i=b(bi); a=(Ci)i=a(ai); k=(Bi)i=k(ki); src=(Ai)i=src(srci). i being the index.

2-Vector instruction _mm_shuffle_epi32

Before answering questions *Q9* and *Q10*, carefully observe and try to understand the figure below which is a screenshot of the SIMDGiraffe prototype, available online at https://github.com/pmntang/SIMDGiraffe. This screenshot is divided into three dials.

On the left dial there is a description of the instruction provided by Intel©.

On the lower dial, there is the graphical translation of this description. This translation consists in displaying for each of these vectors, its fields (or coordinates). We use the letters of the alphabet with subscripts (A0, A1, …B0, B1,…, ...) written inside blue rectangles to designate these fields. The description graphic is preceded on the line, to the left of the equality sign (=), by the name of the vector in question (a, imm8, r) and its type (__m128i, int, __m128i).

On the right-hand dial there is a visual description of the links between each field (or coordinate) of the result vector r and the fields (or coordinates) of the operand vectors used to calculate this field (or this coordinate). This description consists, as we can see, in giving for each field of the vector result r the calculation formula of that field from the operand fields used to carry out this calculation. You can also watch this short video which provides and explanation by an expert in the field of vector programming of this instruction: https://www.youtube.com/watch?v=zSz_-eAI8MA

Q9- After observing the figure above, say what the _mm_shuffle_epi32 instruction does by performing the following calculation: given a=(6, 7, 4, 3); imm8=(0, 1, 2, 3) . Calculate r = _mm_shuffle_epi32(a, imm8) *

◯ r = (6, 2, 1, 3) ◯ r = (6, 7, 4, 3) ◯ r = (3, 4, 7, 6) ◯ r = (3, 7, 4, 6)

Check the radio button in front of the correct answer. You can use the right quadrant of the previous figure as a help.

Q10- Using the figure above again, give a general formula for calculating the coordinates of r(ri) as function of those of a(ai) and imm8(imm8i). ri=? *

- ri=Ci=Aij=aij, where j=Bi=imm8i
- ri=Ci=Ai=ai
- ri=Ci=Ai x Bi=ai x imm8i
- ri=Ci=Aj=aj, where j=Bi=imm8i

Check the radio button in front of the correct answer; pay attention to the fact that on the previous figure r=(Ci)i=r(ri); a=(Ai)i=a(ai); imm8=(Bi)i=imm8(imm8i). i being the index.

B- Preliminary knowledge

I- Knowledge of algebra and vector space

Consider the real vector space R3. For A, B, C, Res1, Res2, five vectors of R3 such that A=(a1, a2, a3), B=(b1, b2, b3), C=(c1, c2, c3), Res1=(x1, x2, x3), Res2=(y1, y2, y3) we define vectSum(A,B,C)=Res1 and vectProd(A,B,C)=Res2 by

$$\begin{cases} x_1 = a_1 - b_1 + c_1 \\ x_2 = a_2 - b_2 + c_2 \\ x_3 = a_3 - b_3 + c_3 \end{cases} \text{ and } \begin{cases} y_1 = b_1 \times (a_1 - c_1) + c_1 \\ y_2 = b_2 \times (a_2 - c_2) + c_2 \\ y_3 = b_3 \times (a_3 - c_3) + c_3 \end{cases}$$

Now let's assume that A=(1, 0,1) ; B=(1,1,0) ; C=(0,1, 1).

Q4- Calculate each of the Res1 and Res2 vectors: Res1= ? Res2=? *

- Res1=(2,1,0); Res2=(1,1,0).
- Res1=(0,0,2); Res2=(1,0,1).
- Res1=(2,2,2); Res2=(1,1,1).
- Res1=(1,0,2); Res2=(0,0,1).

Check the radio button in front of the correct answer

Q5- Give a general formula for calculating the coordinates of Res1(xi) and Res2(yi) as a function of those of A (ai), B (bi) and C (ci). xi=? yi=? *

xi= ; yi= .

Write xi= ; yi= . Then, Write the expression of xi (respectively yi) as function of ai, bi and ci in the space following xi (respectively yi).

II- Knowledge of the C language

Consider the following function f in C: int f (int x, int y) {return x-y;}.

Q6- Choose the two instructions in C (that is, instruction1 and instruction2) which allow you to declare three integer variables a, b, c and to place in c the difference between a and b using the function f. instruction1: ? instruction2: ? *

- ◯ Instruction1: int c, a, b; Instruction2: c=f(a-b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a,b);}
- ◯ Instruction1: int c, a, b; Instruction2: c=f(a,b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a-b);}

Check the radio button in front of the correct answer

D- End time of the questionnaire completion and comments

Before submitting the forms, these questions, in particular questions *Q18* must be filled in.

Q11- End time: *

End time

Fill in this field with the current time, to the nearest minute, when you have finished filling out the questionnaire and if you have completed it without interruption. If you have had interruptions, calculate the end time by deducting the duration of the total interruptions from the current time.

Q12- Other comments and remarks:

Fill in this field with your remarks, comments, and observations on any subject of interest in connection with the study, including the questionnaire, the SIMDGiraffe prototype, etc.

Submit

QuestionnaireGroupe1

Merci de prendre part à cette étude qui vise à comprendre l'influence que peut avoir la visualisation sur la compréhension du comportement de logiciel. Vous avez à répondre à 12 questions dont les réponses vont de remplissages automatiques à quelques calculs que vous pouvez faire mentalement. Vous pouvez aussi vous servir d'une calculatrice ou d'une feuille de calcul Excel. Si vous le souhaitez, vous pouvez répondre de façon anonyme à ce questionnaire en fournissant un pseudonyme à la place de votre nom. Cette étude a obtenu une certification éthique du Comité d'éthique de la recherche avec les êtres humains de l'université TELUQ (CER-TELUQ) numéro 2022-08 du 12 avril 2022.

A- Renseignements personnels, date et heure du début de remplissage du questionnaire

Il est essentiel de commencer par renseigner ces questions avant de passer aux suivantes. Les questions Q2 et Q3 sont automatiquement renseignées par la date courante et l'heure courante respectivement; ainsi, il ne reste que Q1 que vous devez renseigner.

Q1- Nom ou pseudonyme *

Nom ou pseudonyme

Saisissez votre nom dans l'espace ci-dessus, ou votre pseudonyme si vous voulez rester anonyme.

Q2-Date : *

2022-07-25

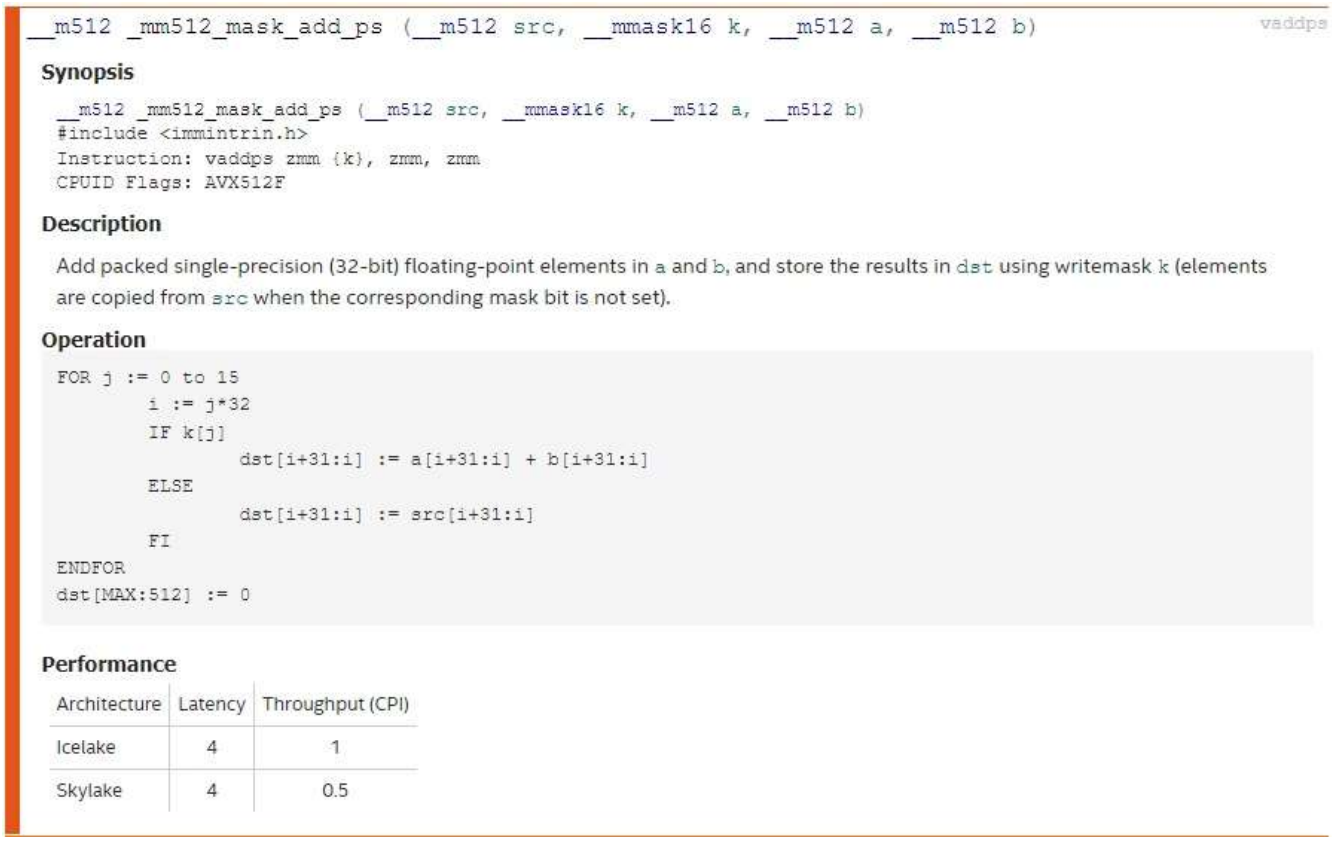
Q3-Heure du début : *

15:40

C- Compréhension du comportement des instructions vectorielles _mm512_mask_add_ps et _mm_shuffle_epi32

1- Instruction vectorielle _mm512_mask_add_ps

Lisez attentivement l'explication de cette instruction donnée sur la figure ci-dessous que vous pouvez retrouver sur le site web d'Intel© avant de répondre aux questions Q7 et Q8. Vous pouvez aussi regarder cette courte vidéo où cette instruction est expliquée.
Vous ne devez utiliser que les explications fournies (vous pouvez naturellement consulter le site web d'Intel©), mais ne faites pas recours à d'autres ressources (par exemple recherche sur Google, autres documents, etc.)



__m512 _mm512_mask_add_ps (__m512 src, __mmask16 k, __m512 a, __m512 b) vaddps

Synopsis

```
__m512 _mm512_mask_add_ps (__m512 src, __mmask16 k, __m512 a, __m512 b)
#include <immintrin.h>
Instruction: vaddps zmm {k}, zmm, zmm
CPUID Flags: AVX512F
```

Description

Add packed single-precision (32-bit) floating-point elements in a and b, and store the results in dst using writemask k (elements are copied from src when the corresponding mask bit is not set).

Operation

```
FOR j := 0 to 15
        i := j*32
        IF k[j]
                dst[i+31:i] := a[i+31:i] + b[i+31:i]
        ELSE
                dst[i+31:i] := src[i+31:i]
        FI
ENDFOR
dst[MAX:512] := 0
```

Performance

Architecture	Latency	Throughput (CPI)
Icelake	4	1
Skylake	4	0.5

Q7- Après avoir bien lu la description et les explications ci-dessus, dites ce que fait l'instruction _mm512_mask_add_ps en effectuant le calcul suivant: étant donnés src=(1, 3, 4, 1, 2, 5, 4, 1, 2, 3, 4, 1, 1, 3, 4, 1) ; k=(1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0) ; a=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) ; b=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) . Calculer r = _ mm512_mask_add_ps(src, k, a, b) *

◯ r = (1, 0, 0, 1, 2, 0, 0, 1, 0, 3, 4, 0, 0, 0, 4, 0) ◯ r = (12, 2, 4, 6, 2, 8, 10, 2, 4, 6, 8, 2, 6, 2, 4, 0)

◯ r = (12, 3, 4, 6, 2, 5, 4, 2, 2, 6, 8, 1, 1, 3, 4, 1) ◯ r = (13, 0, 0, 7, 4, 0, 0, 3, 0, 9, 12, 0, 0, 0, 8, 0)

Cochez le bouton radio devant la bonne réponse.

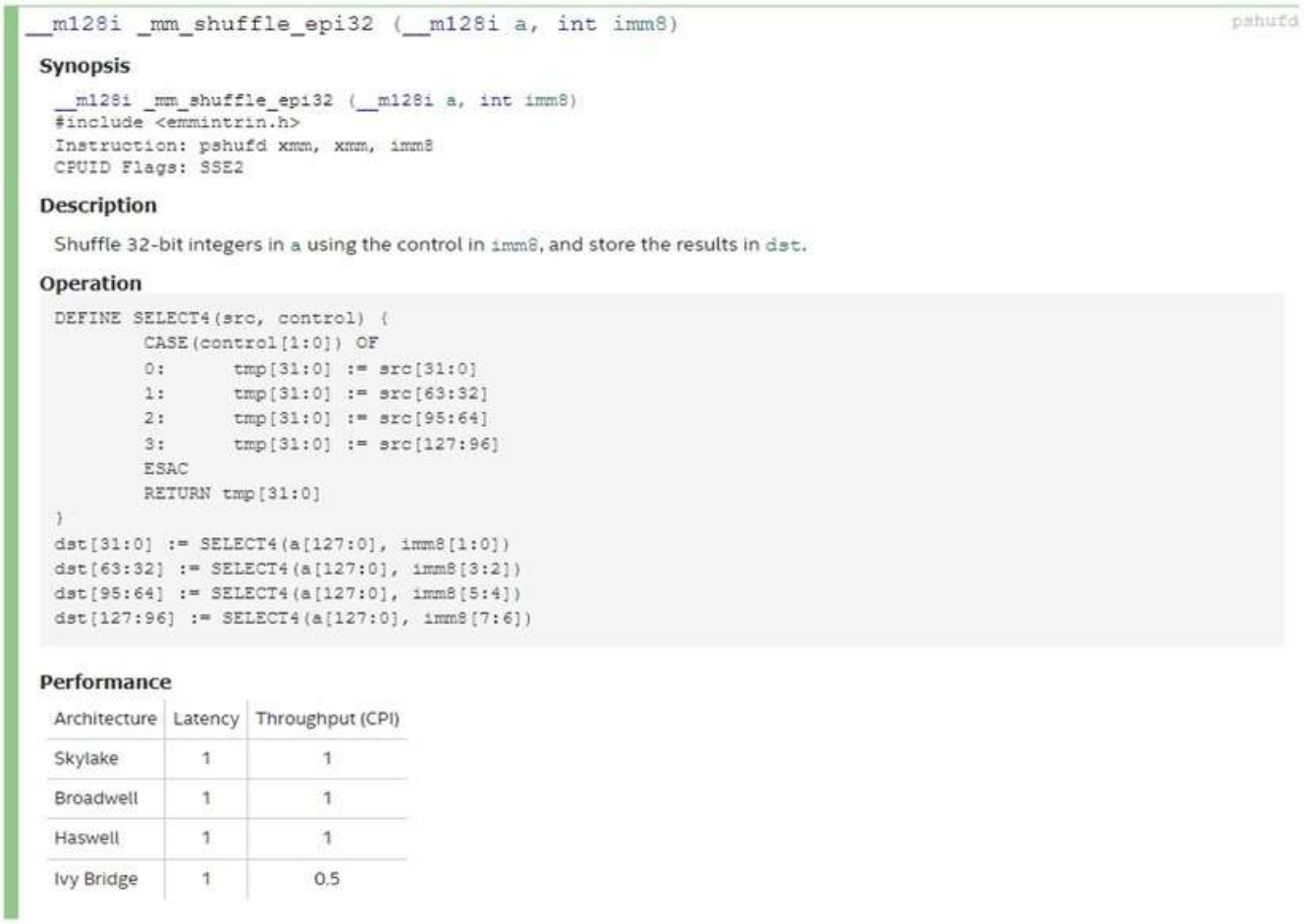
Q8- Toujours à l'aide de la description et des explications ci-dessus, donnez une formule générale de calcul des coordonnées de r(ri) en fonction de celles de src(srci), k(ki), a(ai) et b(bi). ri= ? *

◯ ri=ai+bi ◯ ri=(1-ki) x srci+ai+bi ◯ ri=(1-ki) x srci+ki x (ai+bi) ◯ ri=ki x srci+(1-ki) x (ai+bi)

Cochez le bouton radio devant la bonne réponse. i étant l'indice.

2- Instruction vectorielle _mm_shuffle_epi32

Lisez attentivement l'explication de cette instruction sur la capture d'écran ci-deesous. Vous pouvez retrouver cette explication sur Intel © web site. Assurez-vous autant que possible d'avoir bien lu l'explication avant de répondre aux questions Q9 et Q10. Vous pouvez aussi regarder cette courte vidéo où cette instruction est expliquée ; cette cette courte vidéo est une explication faite par un expert du domaine de la programmation vectorielle. Vous ne devez utiliser que les explications fournies (vous pouvez naturellement consulter le site web d'Intel ©, mais ne faites pas recours à d'autres ressources (par exemple recherche sur Google, autres documents, etc.).



__m128i _mm_shuffle_epi32 (__m128i a, int imm8) pshufd

Synopsis

```
__m128i _mm_shuffle_epi32 (__m128i a, int imm8)
#include <emmintrin.h>
Instruction: pshufd xmm, xmm, imm8
CPUID Flags: SSE2
```

Description

Shuffle 32-bit integers in a using the control in imm8, and store the results in dst.

Operation

```
DEFINE SELECT4(src, control) {
        CASE(control[1:0]) OF
        0:      tmp[31:0] := src[31:0]
        1:      tmp[31:0] := src[63:32]
        2:      tmp[31:0] := src[95:64]
        3:      tmp[31:0] := src[127:96]
        ESAC
        RETURN tmp[31:0]
}
dst[31:0] := SELECT4(a[127:0], imm8[1:0])
dst[63:32] := SELECT4(a[127:0], imm8[3:2])
dst[95:64] := SELECT4(a[127:0], imm8[5:4])
dst[127:96] := SELECT4(a[127:0], imm8[7:6])
```

Performance

Architecture	Latency	Throughput (CPI)
Skylake	1	1
Broadwell	1	1
Haswell	1	1
Ivy Bridge	1	0.5

Q9- Après avoir bien lu la description et les explications ci-dessus, dites ce que fait l'instruction _mm_shuffle_epi32 en effectuant le calcul suivant: étant donnés a=(6, 7, 4, 3) ; imm8=(0, 1, 2, 3) . Calculez r = _mm_shuffle_epi32 (a, imm8) *

◯ r = (6, 2, 1, 3) ◯ r = (6, 7, 4, 3) ◯ r = (3, 4, 7, 6) ◯ r = (3, 7, 4, 6)

Cochez le bouton radio devant la bonne réponse.

Q10- Toujours à l'aide de la description et des explications ci-dessus, donnez une formule générale de calcul des coordonnées de r(ri) en fonction de celles de a(ai) et imm8(imm8i). ri=? *

◯ ri=aij, avec j=imm8i ◯ ri=ai ◯ ri=ai x imm8i ◯ ri=aj, avec j=imm8i

Cochez le bouton radio devant la bonne réponse. i étant l'indice.

B- Connaissances préliminaires

I- Connaissance de l'algèbre et de l'espace vectoriel

Considérons l'espace vectoriel réel R3. Pour A, B, C, Res1, Res2, cinq vecteurs de R3 tels que A=(a1, a2, a3), B=(b1, b2, b3), C=(c1, c2, c3), Res1=(x1, x2, x3), Res2=(y1, y2, y3) on définit vectSum(A,B,C)=Res1 et vectProd(A,B,C)=Res2 par

$$\begin{cases} x_1 = a_1 - b_1 + c_1 \\ x_2 = a_2 - b_2 + c_2 \\ x_3 = a_3 - b_3 + c_3 \end{cases} \text{ and } \begin{cases} y_1 = b_1 \times (a_1 - c_1) + c_1 \\ y_2 = b_2 \times (a_2 - c_2) + c_2 \\ y_3 = b_3 \times (a_3 - c_3) + c_3 \end{cases}$$

On suppose maintenant que A=(1,0,1); B=(1,1,0) ; C=(0,1,1).

Q4- Calculez chacun des vecteurs Res1 et Res2: Res1= ? Res2= ? *

- ◯ Res1=(2,1,0); Res2=(1,1,0).
- ◯ Res1=(0,0,2); Res2=(1,0,1).
- ◯ Res1=(2,2,2); Res2=(1,1,1).
- ◯ Res1=(1,0,2); Res2=(0,0,1).

Cochez le bouton radio devant la bonne réponse

Q5- Donnez une formule générale de calcul des coordonnées de Res1(xi) et de Res2(yi) en fonction de celles de A (ai), B (bi) et C (ci). xi= ? yi= ? *

xi= ; yi= .

Ecrivez xi= ; yi= . Puis, inscrivez l'expression de xi (respectively yi) en fonction des ai, bi et ci dans l'espace devant xi (respectivement yi).

II- Connaissance du langage C

Considérons la fonction f suivante en C: int f (int x, int y) {return x-y;}.

Q6- Déterminez en C deux instructions (soit instruction1 et instruction2) qui permettent de déclarer trois variables entiers a, b, c et de placer dans c la différence de a et b à l'aide de la fonction f. Instruction1: ? Instruction2: ? *

- ◯ Instruction1: int c, a, b; Instruction2: c=f(a-b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a,b);}
- ◯ Instruction1: int c, a, b; Instruction2: c=f(a,b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a-b);}

Cochez le bouton radio devant la bonne réponse

D- Heure de la fin de remplissage du questionnaire et commentaires

Vous avez presque terminé, un dernier effort!

Q11- Heure de fin: *

Heure de fin

Inscrivez dans ce champ l'heure courante, à la minute près, lorsque vous aurez terminé le remplissage du questionnaire et si vous l'avez rempli de manière ininterrompue. Si vous avez marqué des interruptions, calculez l'heure de fin en déduisant de l'heure courante la durée des totale des interruptions.

Q12- Autres commentaires et remarques:

Vos commentaires

Inscrivez dans ce champ vos remarques, commentaires et observations sur tout sujet d'intérêt en rapport avec l'étude dont le questionnaire, le prototype SIMDGiraffe, etc.

Submit

QuestionnaireGroupe2

Merci de prendre part à cette étude qui vise à comprendre l'influence que peut avoir la visualisation sur la compréhension du comportement de logiciel. Vous avez à répondre à 12 questions dont les réponses vont de remplissages automatiques à quelques calculs que vous pouvez faire mentalement. Vous pouvez aussi vous servir d'une calculatrice ou d'une feuille de calcul Excel. Si vous le souhaitez, vous pouvez répondre de façon anonyme à ce questionnaire en fournissant un pseudonyme à la place de votre nom. Cette étude a obtenu une certification éthique du Comité d'éthique de la recherche avec les êtres humains de l'université TELUQ (CER-TELUQ) numéro 2022-08 du 12 avril 2022.

A- Renseignements personnels, date et heure du début de remplissage du questionnaire

Il est essentiel de commencer par renseigner ces questions avant de passer aux suivantes. Les questions Q2 et Q3 sont automatiquement renseignées par la date courante et l'heure courante respectivement; ainsi, il ne reste que Q1 que vous devez renseigner.

Q1- Nom ou pseudonyme *

Nom ou pseudonyme

Saisissez votre nom dans l'espace ci-dessus, ou votre pseudonyme si vous voulez rester anonyme.

Q2-Date : *

2022-07-25

Q3-Heure du début : *

15:39

C- Compréhension du comportement des instructions vectorielles _mm512_mask_add_ps et _mm_shuffle_epi32

1- Instruction vectorielle _mm512_mask_add_ps

Avant de répondre aux questions Q7 et Q8, observez attentivement la figure ci-dessous qui est une capture d'écran du prototype SIMDGiraffe, disponible en ligne sur https://github.com/pmntang/SIMDGiraffe. Cette capture d'écran est divisée en trois cadrans.

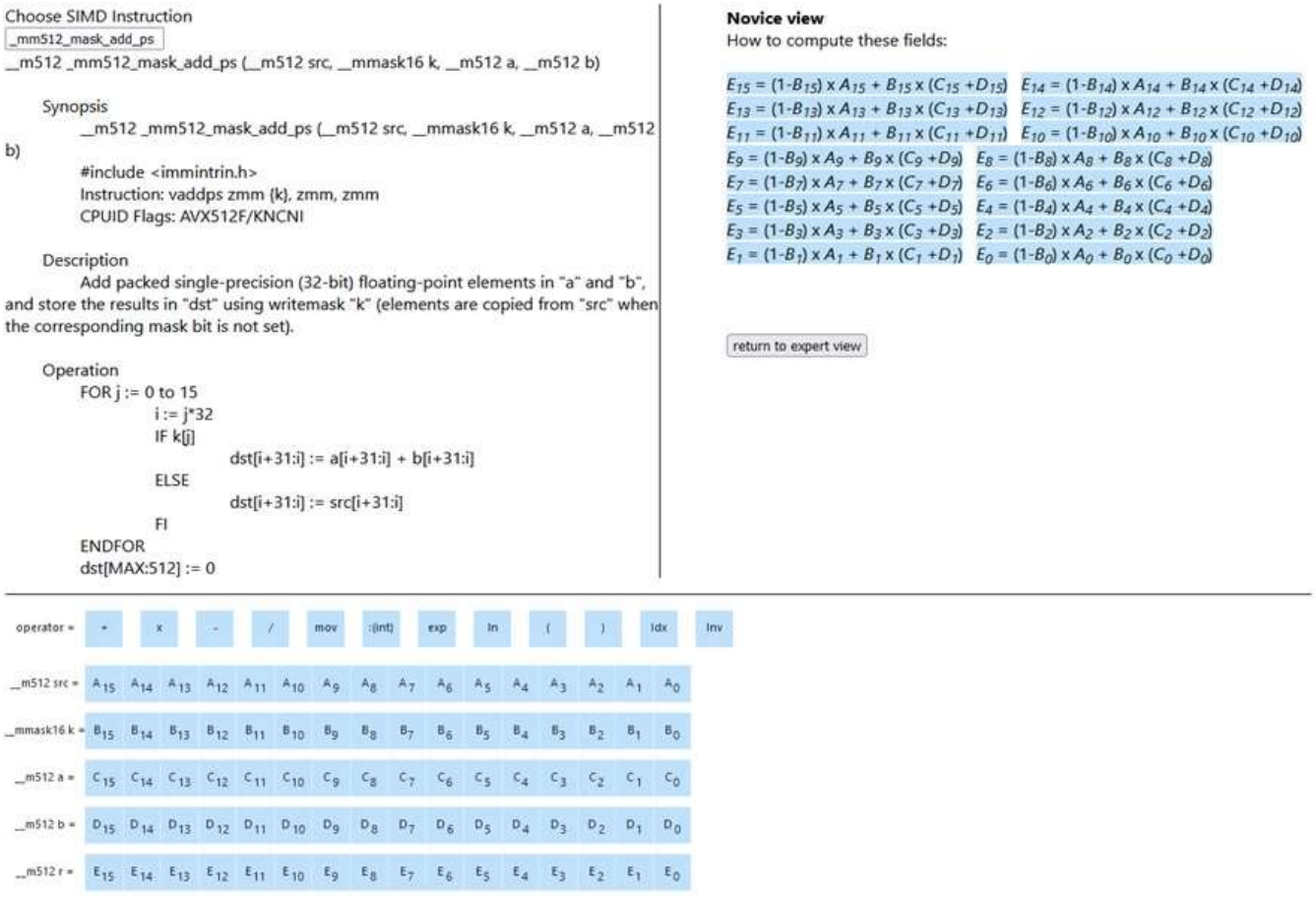
Sur le cadran de gauche il y a une description de l'instruction fournie par Intel©.

Sur le cadran d'en bas, il y a la traduction graphique de cette description. Cette traduction consiste à afficher pour chacun de ces vecteurs, ses champs (ou coordonnées). Nous utilisons les lettres indexées de l'alphabet (A0, A1, …B0, B1,…, ...) inscrites à l'intérieur de rectangles bleus pour désigner ces champs. Par exemple les champs (coordonnées) du vecteur src sont A0, A1, … ce qui veut dire que src0 = A0, src1 = A1… ; ainsi de suite pour les autres vecteurs opérandes ; quant au vecteur résultat, les champs (coordonnées) de r sont E0, E1, … ce qui veut dire que r0 = E0, r1 = E1… La description graphique est précédée sur la ligne, à gauche du signe de l'égalité (=), par le nom du vecteur en question (src, k, a, b, r) et son type (__m512, __mmask16, __m512, __m512, __m512).

Sur le cadran de droite, il y a la description visuelle des liens entre chaque champ (ou coordonnée) du vecteur résultat r et les champs (ou coordonnées) des vecteurs opérandes utilisés pour calculer ce champ (ou cette coordonnée). Cette description consiste comme on peut le remarquer, à donner pour chaque champ du vecteur résultat r la formule qui permet de calculer ce champ à partir des champs opérandes utilisés pour effectuer ce

résultat r la formule qui permet de calculer ce champ à partir des champs opérandes utilisés pour effectuer ce

calcul. Par exemple on peut voir sur ce cadran que r0 = E0 = (1-B0) x A0 + B0 x (C0 +D0), r1 = E1 = (1-B1) x A1 + B1 x (C1 +D1), …
Vous ne devez utiliser que les explications fournies (vous pouvez naturellement consulter le site d'Intel© ou le site de SIMDGiraffe), mais ne faites pas recours à d'autres ressources (par exemple recherche sur Google, autres documents, etc.)



Q7- Après avoir bien observé la figure ci-dessus dite ce que fait l'instruction _mm512_mask_add_ps en effectuant le calcul suivant: étant donnés src=(1, 3, 4, 1, 2, 5, 4, 1, 2, 3, 4, 1, 1, 3, 4, 1) ; k=(1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0) ; a=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) ; b=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) . Calculer r = _mm512_mask_add_ps(src, k, a, b) *

◯ r = (1, 0, 0, 1, 2, 0, 0, 1, 0, 3, 4, 0, 0, 0, 4, 0) ◯ r = (12, 2, 4, 6, 2, 8, 10, 2, 4, 6, 8, 2, 6, 2, 4, 0)

◯ r = (12, 3, 4, 6, 2, 5, 4, 2, 2, 6, 8, 1, 1, 3, 4, 1) ◯ r = (13, 0, 0, 7, 4, 0, 0, 3, 0, 9, 12, 0, 0, 0, 8, 0)

Cochez le bouton radio devant la bonne réponse. Vous pouvez vous aider du quadrant de droite de la figure précédente

Q8- Toujours à l'aide de la figure ci-dessus donnez une formule générale de calcul des coordonnées de r(ri) en fonction de celles de src(srci), k(ki), a(ai) et b(bi). ri= ? *

◯ ri=Ei=Ci+Di=ai+bi ◯ ri=Ei=(1-Bi) x Ai+Ci+Di=(1-ki) x srci+ai+bi

◯ ri=Ei=(1-Bi) x Ai+Bi x (Ci+Di)=(1-ki) x srci+ki x (ai+bi) ◯ ri=Ei=Bi x Ai+(1-Bi) x (Ci+Di)=ki x srci+(1-ki) x (ai+bi)

Cochez le bouton radio devant la bonne réponse en remarquant que sur la figure précédente r=(Ei)i=r(ri); b=(Di)i=b(bi); a=(Ci)i=a(ai); k=(Bi)i=k(ki); src=(Ai)i=src(srci). i étant l'indice.

2- Instruction vectorielle _mm_shuffle_epi32

Avant de répondre aux questions Q9 et Q10, observez attentivement la figure ci-dessous qui est une capture d'écran du prototype SIMDGiraffe, disponible en ligne sur https://github.com/pmntang/SIMDGiraffe. Cette capture d'écran est divisée en trois cadrans.

Sur le cadran de gauche il y a une description de l'instruction fournie par Intel©.

Sur le cadran d'en bas, il y a la traduction graphique de cette description. Cette traduction consiste à afficher pour chacun de ces vecteurs, ses champs (ou coordonnées). Nous utilisons les lettres indexées de l'alphabet (A0, A1, …B0, B1,…, ...) inscrites à l'intérieur de rectangles bleus pour désigner ces champs. Ainsi les champs (coordonnées) du vecteur a sont A0, A1, A2, A3, ce qui veut dire que a0 = A0, a1 = A1, a2 = A2, a3 = A3 ; les champs (coordonnées) du vecteur imm8 sont B0, B1, B2, B3, ce qui veut dire que imm80 = B0, imm81 = B1, imm82 = B2, imm83 = B3 ; quant au vecteur résultat, les champs (coordonnées) de r sont C0, C1, C2, C3; ce qui veut dire que r0 = C0, r1 = C1, r2 = C2, r3 = C3. La description graphique est précédée sur la ligne, à gauche du signe de l'égalité (=), par le nom du vecteur en question (a, imm8, r) et son type (__m128i, int, __m128i).

Sur le cadran de droite, il y a la description visuelle des liens entre chaque champ (ou coordonnée) du vecteur résultat r et les champs (ou coordonnées) des vecteurs opérandes utilisés pour calculer ce champ (ou cette coordonnée). Cette description consiste comme on peut le remarquer, à donner pour chaque champ du vecteur résultat r la formule qui permet de calculer ce champ à partir des champs opérandes utilisés pour effectuer ce calcul. On peut ainsi voir sur ce cadran que r0= C0 = AB0; r1 = C1 = AB1; r2 = C2 = AB2; r3 = C3 = AB3.
Vous ne devez utiliser que les explications fournies (vous pouvez naturellement consulter le site d'Intel© ou le site de SIMDGiraffe), mais ne faites pas recours à d'autres ressources (par exemple recherche sur Google, autres documents, etc.).

Q9- Après avoir bien observé la figure ci-dessus dite ce que fait l'instruction _mm_shuffle_epi32 en effectuant le calcul suivant: étant donnés a=(6, 7, 4, 3) ; imm8=(0, 1, 2, 3) . Calculez r = _mm_shuffle_epi32 (a, imm8) *

◯ r = (6, 2, 1, 3) ◯ r = (6, 7, 4, 3) ◯ r = (3, 4, 7, 6) ◯ r = (3, 7, 4, 6)

Cochez le bouton radio devant la bonne réponse. Vous pouvez vous aider du quadrant de droite de la figure précédente

Q10- Toujours à l'aide de la figure ci-dessus, donnez une formule générale de calcul des coordonnées de r(ri) en fonction de celles de a(ai) et imm8(imm8i). ri=? *

◯ ri=Ci=Aij=aij, avec j=Bi=imm8i ◯ ri=Ci=Ai=ai ◯ ri=Ci=Ai x Bi=ai x imm8i ◯ ri=Ci=Aj=aj, avec j=Bi=imm8i

Cochez le bouton radio devant la bonne réponse en remarquant que sur la figure précédente r=(Ci)i=r(ri); a=(Ai)i=a(ai); imm8=(Bi)i=imm8(imm8i). i étant l'indice.

B- Connaissances préliminaires

I- Connaissance de l'algèbre et de l'espace vectoriel

Considérons l'espace vectoriel réel R3. Pour A, B, C, Res1, Res2, cinq vecteurs de R3 tels que A=(a1, a2, a3), B=(b1, b2, b3), C=(c1, c2, c3), Res1=(x1, x2, x3), Res2=(y1, y2, y3) on définit vectSum(A,B,C)=Res1 et vectProd(A,B,C)=Res2 par

$$\begin{cases} x_1 = a_1 - b_1 + c_1 \\ x_2 = a_2 - b_2 + c_2 \\ x_3 = a_3 - b_3 + c_3 \end{cases} \text{ and } \begin{cases} y_1 = b_1 \times (a_1 - c_1) + c_1 \\ y_2 = b_2 \times (a_2 - c_2) + c_2 \\ y_3 = b_3 \times (a_3 - c_3) + c_3 \end{cases}$$

On suppose maintenant que A=(1,0,1); B=(1,1,0) ; C=(0,1,1).

Q4- Calculez chacun des vecteurs Res1 et Res2: Res1= ? Res2= ? *

- ◯ Res1=(2,1,0); Res2=(1,1,0).
- ◯ Res1=(0,0,2); Res2=(1,0,1).
- ◯ Res1=(2,2,2); Res2=(1,1,1).
- ◯ Res1=(1,0,2); Res2=(0,0,1).

Cochez le bouton radio devant la bonne réponse

Q5- Donnez une formule générale de calcul des coordonnées de Res1(xi) et de Res2(yi) en fonction de celles de A (ai), B (bi) et C (ci). xi= ? yi= ? *

xi= ; yi= .

Ecrivez xi= ; yi= . Puis, inscrivez l'expression de xi (respectively yi) en fonction des ai, bi et ci dans l'espace devant xi (respectivement yi).

II- Connaissance du langage C

Considérons la fonction f suivante en C: int f (int x, int y) {return x-y;}.

Q6- Déterminez en C deux instructions (soit instruction1 et instruction2) qui permettent de déclarer trois variables entiers a, b, c et de placer dans c la différence de a et b à l'aide de la fonction f. Instruction1: ? Instruction2: ? *

- ◯ Instruction1: int c, a, b; Instruction2: c=f(a-b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a,b);}
- ◯ Instruction1: int c, a, b; Instruction2: c=f(a,b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a-b);}

Cochez le bouton radio devant la bonne réponse

D- Heure de la fin de remplissage du questionnaire et commentaires

Vous avez presque terminé, un dernier effort!

Q11- Heure de fin: *

Heure de fin

Inscrivez dans ce champ l'heure courante, à la minute près, lorsque vous aurez terminé le remplissage du questionnaire et si vous l'avez rempli de manière ininterrompue. Si vous avez marqué des interruptions, calculez l'heure de fin en déduisant de l'heure courante la durée des totale des interruptions.

Q12- Autres commentaires et remarques:

Vos commentaires

Inscrivez dans ce champ vos remarques, commentaires et observations sur tout sujet d'intérêt en rapport avec l'étude dont le questionnaire, le prototype SIMDGiraffe, etc.

Submit

QuestionnaireGroupe3

Merci de prendre part à cette étude qui vise à comprendre l'influence que peut avoir la visualisation sur la compréhension du comportement de logiciel. Vous avez à répondre à 12 questions dont les réponses vont de remplissages automatiques à quelques calculs que vous pouvez faire mentalement. Vous pouvez aussi vous servir d'une calculatrice ou d'une feuille de calcul Excel. Si vous le souhaitez, vous pouvez répondre de façon anonyme à ce questionnaire en fournissant un pseudonyme à la place de votre nom. Cette étude a obtenu une certification éthique du Comité d'éthique de la recherche avec les êtres humains de l'université TELUQ (CER-TELUQ) numéro 2022-08 du 12 avril 2022.

A- Renseignements personnels, date et heure du début de remplissage du questionnaire

Il est essentiel de commencer par renseigner ces questions avant de passer aux suivantes. Les questions Q2 et Q3 sont automatiquement renseignées par la date courante et l'heure courante respectivement; ainsi, il ne reste que Q1 que vous devez renseigner.

Q1- Nom ou pseudonyme *

Nom ou pseudonyme

Saisissez votre nom dans l'espace ci-dessus, ou votre pseudonyme si vous voulez rester anonyme.

Q2-Date : *

2022-07-25

Q3-Heure du début : *

15:38

C- Compréhension du comportement des instructions vectorielles _mm512_mask_add_ps et _mm_shuffle_epi32

1- Instruction vectorielle _mm512_mask_add_ps

Avant de répondre aux questions Q7 et Q8, observez attentivement la figure ci-dessous qui est une capture d'écran du prototype SIMDGiraffe, disponible en ligne sur https://github.com/pmntang/SIMDGiraffe. Cette capture d'écran est divisée en trois cadrans.

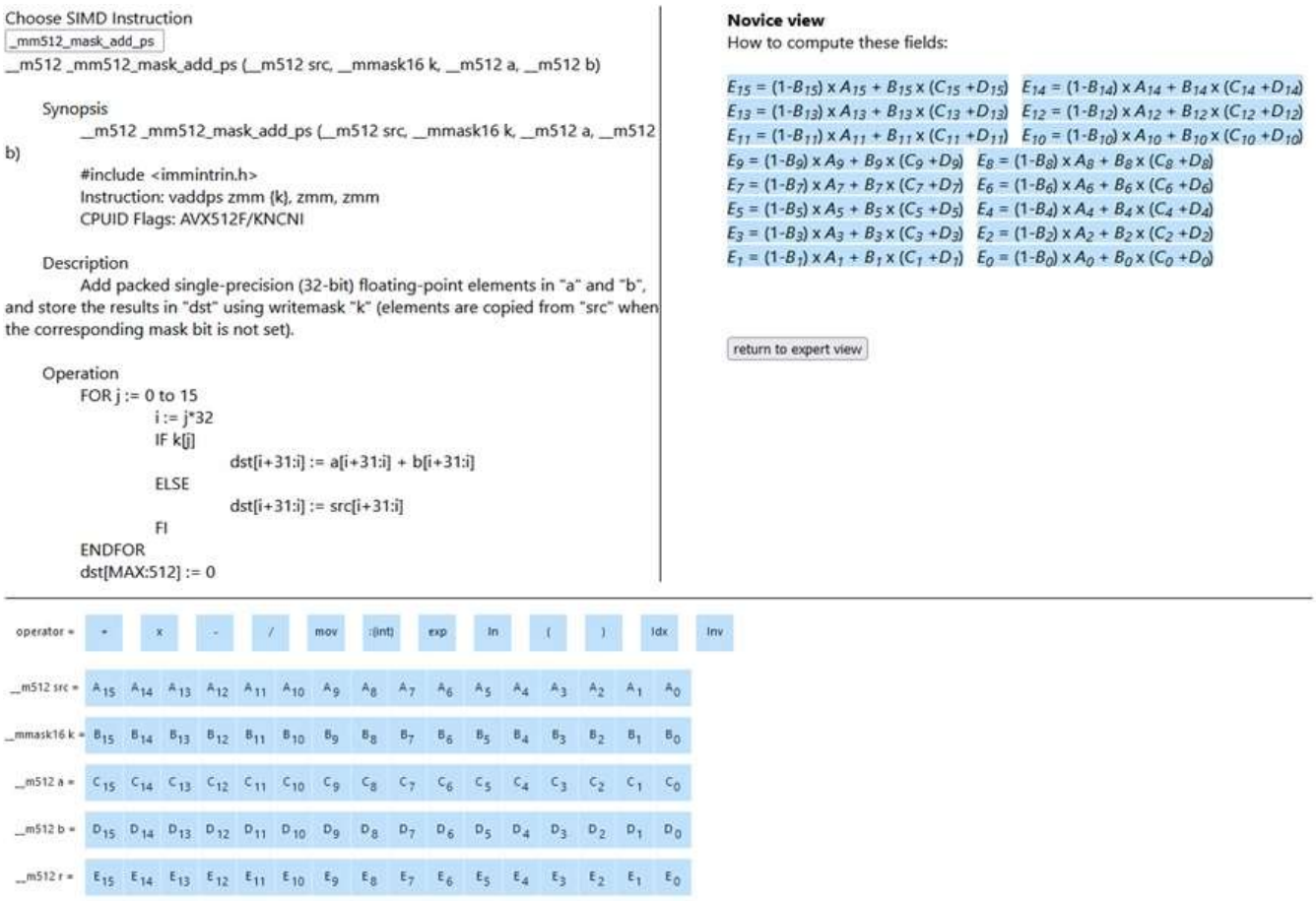
Sur le cadran de gauche il y a une description de l'instruction fournie par Intel©.

Sur le cadran d'en bas, il y a la traduction graphique de cette description. Cette traduction consiste à afficher pour chacun de ces vecteurs, ses champs (ou coordonnées). Nous utilisons les lettres indexées de l'alphabet (A0, A1, …B0, B1,…, ...) inscrites à l'intérieur de rectangles bleus pour désigner ces champs. Par exemple les champs (coordonnées) du vecteur src sont A0, A1, … ce qui veut dire que src0 = A0, src1 = A1… ; ainsi de suite pour les autres vecteurs opérandes ; quant au vecteur résultat, les champs (coordonnées) de r sont E0, E1, … ce qui veut dire que r0 = E0, r1 = E1… La description graphique est précédée sur la ligne, à gauche du signe de l'égalité (=), par le nom du vecteur en question (src, k, a, b, r) et son type (__m512, __mmask16, __m512, __m512, __m512).

Sur le cadran de droite, il y a la description visuelle des liens entre chaque champ (ou coordonnée) du vecteur résultat r et les champs (ou coordonnées) des vecteurs opérandes utilisés pour calculer ce champ (ou cette coordonnée). Cette description consiste comme on peut le remarquer, à donner pour chaque champ du vecteur résultat r la formule qui permet de calculer ce champ à partir des champs opérandes utilisés pour effectuer ce

résultat r la formule qui permet de calculer ce champ à partir des champs opérandes utilisés pour effectuer ce

calcul. Par exemple on peut voir sur ce cadran que r0 = E0 = (1-B0) x A0 + B0 x (C0 +D0), r1 = E1 = (1-B1) x A1 + B1 x (C1 +D1), …
Vous ne devez utiliser que les explications fournies (vous pouvez naturellement consulter le site d'Intel© ou le site de SIMDGiraffe), mais ne faites pas recours à d'autres ressources (par exemple recherche sur Google, autres documents, etc.). Vous pouvez aussi regarder cette
courte vidéo où cette instruction est expliquée par un expert du domaine de la programmation vectorielle: https://youtu.be/omQ0ebeYJBg



Q7- Après avoir bien observé la figure ci-dessus, dites ce que fait l'instruction _mm512_mask_add_ps en effectuant le calcul suivant: étant donnés src=(1, 3, 4, 1, 2, 5, 4, 1, 2, 3, 4, 1, 1, 3, 4, 1) ; k=(1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0) ; a=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) ; b=(6, 1, 2, 3, 1, 4, 5, 1, 2, 3, 4, 1, 3,1, 2, 1) . Calculer r = _mm512_mask_add_ps(src, k, a, b) *

- ◯ r = (1, 0, 0, 1, 2, 0, 0, 1, 0, 3, 4, 0, 0, 0, 4, 0)
- ◯ r = (12, 2, 4, 6, 2, 8, 10, 2, 4, 6, 8, 2, 6, 2, 4, 0)
- ◯ r = (12, 3, 4, 6, 2, 5, 4, 2, 2, 6, 8, 1, 1, 3, 4, 1)
- ◯ r = (13, 0, 0, 7, 4, 0, 0, 3, 0, 9, 12, 0, 0, 0, 8, 0)

Cochez le bouton radio devant la bonne réponse. Vous pouvez vous aider du quadrant de droite de la figure précédente

Q8- Toujours à l'aide de la figure ci-dessus, donnez une formule générale de calcul des coordonnées de r(ri) en fonction de celles de src(srci), k(ki), a(ai) et b(bi). ri= ? *

- ◯ ri=Ei=Ci+Di=ai+bi
- ◯ ri=Ei=(1-Bi) x Ai+Ci+Di=(1-ki) x srci+ai+bi
- ◯ ri=Ei=(1-Bi) x Ai+Bi x (Ci+Di)=(1-ki) x srci+ki x (ai+bi)
- ◯ ri=Ei=Bi x Ai+(1-Bi) x (Ci+Di)=ki x srci+(1-ki) x (ai+bi)

Cochez le bouton radio devant la bonne réponse en remarquant que sur la figure précédente r=(Ei)i=r(ri); b=(Di)i=b(bi); a=(Ci)i=a(ai); k=(Bi)i=k(ki); src=(Ai)i=src(srci). i étant l'indice.

2- Instruction vectorielle _mm_shuffle_epi32

Avant de répondre aux questions Q9 et Q10, observez attentivement la figure ci-dessous qui est une capture d'écran du prototype SIMDGiraffe, disponible en ligne sur https://github.com/pmntang/SIMDGiraffe. Cette capture d'écran est divisée en trois cadrans.

Sur le cadran de gauche il y a une description de l'instruction fournie par Intel©.

Sur le cadran d'en bas, il y a la traduction graphique de cette description. Cette traduction consiste à afficher pour chacun de ces vecteurs, ses champs (ou coordonnées). Nous utilisons les lettres indexées de l'alphabet (A0, A1, …B0, B1,…, ...) inscrites à l'intérieur de rectangles bleus pour désigner ces champs. Ainsi les champs (coordonnées) du vecteur a sont A0, A1, A2, A3, ce qui veut dire que a0 = A0, a1 = A1, a2 = A2, a3 = A3 ; les champs (coordonnées) du vecteur imm8 sont B0, B1, B2, B3, ce qui veut dire que imm80 = B0, imm81 = B1, imm82 = B2, imm83 = B3 ; quant au vecteur résultat, les champs (coordonnées) de r sont C0, C1, C2, C3; ce qui veut dire que r0 = C0, r1 = C1, r2 = C2, r3 = C3. La description graphique est précédée sur la ligne, à gauche du signe de l'égalité (=), par le nom du vecteur en question (a, imm8, r) et son type (__m128i, int, __m128i).

Sur le cadran de droite, il y a la description visuelle des liens entre chaque champ (ou coordonnée) du vecteur résultat r et les champs (ou coordonnées) des vecteurs opérandes utilisés pour calculer ce champ (ou cette coordonnée). Cette description consiste comme on peut le remarquer, à donner pour chaque champ du vecteur résultat r la formule qui permet de calculer ce champ à partir des champs opérandes utilisés pour effectuer ce calcul. On peut ainsi voir sur ce cadran que r0= C0 = AB0 r1 = C1 = AB1 r2 = C2 = AB2 r3 = C3 = AB3.
Vous ne devez utiliser que les explications fournies (vous pouvez naturellement consulter le site d'Intel© ou le site de SIMDGiraffe), mais ne faites pas recours à d'autres ressources (par exemple recherche sur Google, autres documents, etc.). Vous pouvez aussi regarder cette courte vidéo où cette instruction est expliquée par un expert du domaine de la programmation vectorielle: https://www.youtube.com/watch?v=WVz1jHTIOtY

Q9- Après avoir bien observé la figure ci-dessus, dites ce que fait l'instruction _mm_shuffle_epi32 en effectuant le calcul suivant: étant donnés a=(6, 7, 4, 3) ; imm8=(0, 1, 2, 3) . Calculez r = _mm_shuffle_epi32 (a, imm8) *

◯ r = (6, 2, 1, 3) ◯ r = (6, 7, 4, 3) ◯ r = (3, 4, 7, 6) ◯ r = (3, 7, 4, 6)

Cochez le bouton radio devant la bonne réponse. Vous pouvez vous aider du quadrant de droite de la figure précédente

Q10- Toujours à l'aide de la figure ci-dessus, donnez une formule générale de calcul des coordonnées de r(ri) en fonction de celles de a(ai) et imm8(imm8i). ri=? *

◯ ri=Ci=Aij=aij, avec j=Bi=imm8i ◯ ri=Ci=Ai=ai ◯ ri=Ci=Ai x Bi=ai x imm8i ◯ ri=Ci=Aj=aj, avec j=Bi=imm8i

Cochez le bouton radio devant la bonne réponse en remarquant que sur la figure précédente r=(Ci)i=r(ri); a=(Ai)i=a(ai); imm8=(Bi)i=imm8(imm8i). i étant l'indice.

B- Connaissances préliminaires

I- Connaissance de l'algèbre et de l'espace vectoriel

Considérons l'espace vectoriel réel R3. Pour A, B, C, Res1, Res2, cinq vecteurs de R3 tels que A=(a1, a2, a3), B=(b1, b2, b3), C=(c1, c2, c3), Res1=(x1, x2, x3), Res2=(y1, y2, y3) on définit vectSum(A,B,C)=Res1 et vectProd(A,B,C)=Res2 par

$$\begin{cases} x_1 = a_1 - b_1 + c_1 \\ x_2 = a_2 - b_2 + c_2 \\ x_3 = a_3 - b_3 + c_3 \end{cases} \text{ and } \begin{cases} y_1 = b_1 \times (a_1 - c_1) + c_1 \\ y_2 = b_2 \times (a_2 - c_2) + c_2 \\ y_3 = b_3 \times (a_3 - c_3) + c_3 \end{cases}$$

On suppose maintenant que A=(1,0,1); B=(1,1,0) ; C=(0,1,1).

Q4- Calculez chacun des vecteurs Res1 et Res2: Res1= ? Res2= ? *

- ◯ Res1=(2,1,0); Res2=(1,1,0).
- ◯ Res1=(0,0,2); Res2=(1,0,1).
- ◯ Res1=(2,2,2); Res2=(1,1,1).
- ◯ Res1=(1,0,2); Res2=(0,0,1).

Cochez le bouton radio devant la bonne réponse

Q5- Donnez une formule générale de calcul des coordonnées de Res1(xi) et de Res2(yi) en fonction de celles de A (ai), B (bi) et C (ci). xi= ? yi= ? *

xi= ; yi= .

Ecrivez xi= ; yi= . Puis, inscrivez l'expression de xi (respectively yi) en fonction des ai, bi et ci dans l'espace devant xi (respectivement yi).

II- Connaissance du langage C

Considérons la fonction f suivante en C: int f (int x, int y) {return x-y;}.

Q6- Déterminez en C deux instructions (soit instruction1 et instruction2) qui permettent de déclarer trois variables entiers a, b, c et de placer dans c la différence de a et b à l'aide de la fonction f. Instruction1: ? Instruction2: ? *

- ◯ Instruction1: int c, a, b; Instruction2: c=f(a-b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a,b);}
- ◯ Instruction1: int c, a, b; Instruction2: c=f(a,b);
- ◯ Instruction1: int c, a, b; Instruction2: {return c=f(a-b);}

Cochez le bouton radio devant la bonne réponse

D- Heure de la fin de remplissage du questionnaire et commentaires

Vous avez presque terminé, un dernier effort!

Q18- Heure de fin: *

Heure de fin

Inscrivez dans ce champ l'heure courante, à la minute près, lorsque vous aurez terminé le remplissage du questionnaire et si vous l'avez rempli de manière ininterrompue. Si vous avez marqué des interruptions, calculez l'heure de fin en déduisant de l'heure courante la durée des totale des interruptions.

Q19- Autres commentaires et remarques:

Vos commentaires

Inscrivez dans ce champ vos remarques, commentaires et observations sur tout sujet d'intérêt en rapport avec l'étude dont le questionnaire, le prototype SIMDGiraffe, etc.

Submit

APPENDICE C

LES LETTRES DE SOLLICITATION ENVOYÉES

Invitation

Dear Sir/Madam,

We are soliciting participants for the study entitled: "SIMDGiraffe: Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions" in the aim to understand the influence and importance of visualization on understanding computer code behavior.

Objective of the study: To verify if the visualization tool prototype, SIMDGiraffe, that we develop, and which is available online at https://github.com/pmntang/SIMDGiraffe helps a novice in the field of vector instructions to understand the behavior of these instructions.

If you are interested, we kindly ask you to fill out the online questionnaire whose link in the **blue** follows at the end of this message. You can fill out this questionnaire anonymously by entering a pseudonym of your choice in the field “name or pseudonym”. If you enter your name in this field, your name will not be used in the analyses but will simply be transformed into a non-reversible key. None of your personal information is therefore stored.

Characteristic of the study population:

- Can express themselves in English;

- Has knowledge-even very rudimentary knowledge-of vector spaces or programming in a language like C;

- Is at least 17 years old.

If you don't want to participate in this study, you can just ignore this email.

This research has received ethics certification from the Ethics Committee for Research Involving Humans (CER-TELUQ) on 2022/04/12. Reference no: 2022-08.

Do you have questions about this study? Write to us at ntang.pierre_marie@univ.teluq.ca

The masculine gender is used here just for simplicity. Our study does not discriminate based on gender.

Click on the link to fill out the questionnaire or copy and paste into your browser the following url https://www.cognitoforms.com/PierreMarieNtang/QuestionnaireGroup1

French version to follow / La version en français va suivre.

Madame, Monsieur,

Nous sollicitons des participants à l'étude intitulée : « SIMDGiraffe : Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions » dans le cadre de la compréhension de l'influence et de l'importance de la visualisation sur la compréhension du comportement de code informatique.

Objectif de l'étude : Vérifier si le prototype d'outil de visualisation SIMDGiraffe que nous avons développé, et qui est disponible en ligne à https://github.com/pmntang/SIMDGiraffe aide un novice du domaine des instructions vectorielles à comprendre le comportement de ces instructions.

Si vous êtes intéressés, nous vous prions de bien vouloir remplir le questionnaire en ligne dont le lien en **vert** suit à la fin de ce message. Vous pouvez remplir ce questionnaire de façon anonyme en inscrivant un pseudonyme de votre choix dans le champ « nom ou pseudonyme ». Si vous inscrivez votre nom dans ce champ, votre nom ne sera pas utilisé dans les analyses mais ; il sera simplement transformé en une clef non réversible. Aucune de vos informations personnelles n'est donc stockée.

Caractéristique de la population à l'étude :

– Peut s'exprimer en français;

– A des connaissances-même des connaissances très rudimentaires-en espaces vectoriels ou en programmation dans un langage comme le C ;

– Avoir au moins 17 ans.

Si vous ne voulez pas participer à cette étude, vous pouvez juste ignorer ce courriel.

Cette recherche a obtenu une certification éthique du Comité d'éthique de la recherche avec les êtres humains (CER-TELUQ) le 12/04/2022. No de référence : 2022-08.

Vous avez des questions au sujet de cette étude ? Écrivez-nous à ntang.pierre_marie@univ.teluq.ca

Le masculin est utilisé ici juste à titre de simplification. Notre étude ne fait aucune discrimination basée sur le genre.

Cliquez sur le lien pour remplir le questionnaire ou copier et coller l'url qui suit dans votre navigateur https://www.cognitoforms.com/PierreMarieNtang/QuestionnaireGroupe1

Invitation

Dear Sir/Madam,

We are soliciting participants for the study entitled: "SIMDGiraffe: Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions" in the aim to understand the influence and importance of visualization on understanding computer code behavior.

Objective of the study: To verify if the visualization tool prototype, SIMDGiraffe, that we develop, and which is available online at https://github.com/pmntang/SIMDGiraffe helps a novice in the field of vector instructions to understand the behavior of these instructions.

If you are interested, we kindly ask you to fill out the online questionnaire whose link in the **blue** follows at the end of this message. You can fill out this questionnaire anonymously by entering a pseudonym of your choice in the field “name or pseudonym”. If you enter your name in this field, your name will not be used in the analyses but will simply be transformed into a non-reversible key. None of your personal information is therefore stored.

Characteristic of the study population:

- Can express themselves in English;

- Has knowledge-even very rudimentary knowledge-of vector spaces or programming in a language like C;

- Is at least 17 years old.

If you don't want to participate in this study, you can just ignore this email.

This research has received ethics certification from the Ethics Committee for Research Involving Humans (CER-TELUQ) on 2022/04/12. Reference no: 2022-08.

Do you have questions about this study? Write to us at ntang.pierre_marie@univ.teluq.ca

The masculine gender is used here just for simplicity. Our study does not discriminate based on gender.

Click on the link to fill out the questionnaire or copy and paste into your browser the following url https://www.cognitoforms.com/PierreMarieNtang/QuestionnaireGroup2

French version to follow / La version en français va suivre.

Madame, Monsieur,

Nous sollicitons des participants à l'étude intitulée : « SIMDGiraffe : Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions » dans le cadre de la compréhension de l'influence et de l'importance de la visualisation sur la compréhension du comportement de code informatique.

Objectif de l'étude : Vérifier si le prototype d'outil de visualisation SIMDGiraffe que nous avons développé, et qui est disponible en ligne à https://github.com/pmntang/SIMDGiraffe aide un novice du domaine des instructions vectorielles à comprendre le comportement de ces instructions.

Si vous êtes intéressés, nous vous prions de bien vouloir remplir le questionnaire en ligne dont le lien en **vert** suit à la fin de ce message. Vous pouvez remplir ce questionnaire de façon anonyme en inscrivant un pseudonyme de votre choix dans le champ « nom ou pseudonyme ». Si vous inscrivez votre nom dans ce champ, votre nom ne sera pas utilisé dans les analyses mais ; il sera simplement transformé en une clef non réversible. Aucune de vos informations personnelles n'est donc stockée.

Caractéristique de la population à l'étude :

– Peut s'exprimer en français;

– A des connaissances-même des connaissances très rudimentaires-en espaces vectoriels ou en programmation dans un langage comme le C ;

– Avoir au moins 17 ans.

Si vous ne voulez pas participer à cette étude, vous pouvez juste ignorer ce courriel.

Cette recherche a obtenu une certification éthique du Comité d'éthique de la recherche avec les êtres humains (CER-TELUQ) le 12/04/2022. No de référence : 2022-08.

Vous avez des questions au sujet de cette étude ? Écrivez-nous à ntang.pierre_marie@univ.teluq.ca

Le masculin est utilisé ici juste à titre de simplification. Notre étude ne fait aucune discrimination basée sur le genre.

Cliquez sur le lien pour remplir le questionnaire ou copier et coller l'url qui suit dans votre navigateur https://www.cognitoforms.com/PierreMarieNtang/QuestionnaireGroupe2

Invitation

Dear Sir/Madam,

We are soliciting participants for the study entitled: "SIMDGiraffe: Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions" in the aim to understand the influence and importance of visualization on understanding computer code behavior.

Objective of the study: To verify if the visualization tool prototype, SIMDGiraffe, that we develop, and which is available online at https://github.com/pmntang/SIMDGiraffe helps a novice in the field of vector instructions to understand the behavior of these instructions.

If you are interested, we kindly ask you to fill out the online questionnaire whose link in the **blue** follows at the end of this message. You can fill out this questionnaire anonymously by entering a pseudonym of your choice in the field “name or pseudonym”. If you enter your name in this field, your name will not be used in the analyses but will simply be transformed into a non-reversible key. None of your personal information is therefore stored.

Characteristic of the study population:

- Can express themselves in English;

- Has knowledge-even very rudimentary knowledge-of vector spaces or programming in a language like C;

- Is at least 17 years old.

If you don't want to participate in this study, you can just ignore this email.

This research has received ethics certification from the Ethics Committee for Research Involving Humans (CER-TELUQ) on 2022/04/12. Reference no: 2022-08.

Do you have questions about this study? Write to us at ntang.pierre_marie@univ.teluq.ca

The masculine gender is used here just for simplicity. Our study does not discriminate based on gender.

Click on the link to fill out the questionnaire or copy and paste into your browser the following url https://www.cognitoforms.com/PierreMarieNtang/QuestionnaireGroup3

French version to follow / La version en français va suivre.

Madame, Monsieur,

Nous sollicitons des participants à l'étude intitulée : « SIMDGiraffe : Capturing and Visualizing the Expert's Mind to Understand SIMD Instructions » dans le cadre de la compréhension de l'influence et de l'importance de la visualisation sur la compréhension du comportement de code informatique.

Objectif de l'étude : Vérifier si le prototype d'outil de visualisation SIMDGiraffe que nous avons développé, et qui est disponible en ligne à https://github.com/pmntang/SIMDGiraffe aide un novice du domaine des instructions vectorielles à comprendre le comportement de ces instructions.

Si vous êtes intéressés, nous vous prions de bien vouloir remplir le questionnaire en ligne dont le lien en **vert** suit à la fin de ce message. Vous pouvez remplir ce questionnaire de façon anonyme en inscrivant un pseudonyme de votre choix dans le champ « nom ou pseudonyme ». Si vous inscrivez votre nom dans ce champ, votre nom ne sera pas utilisé dans les analyses mais ; il sera simplement transformé en une clef non réversible. Aucune de vos informations personnelles n'est donc stockée.

Caractéristique de la population à l'étude :

– Peut s'exprimer en français;

– A des connaissances-même des connaissances très rudimentaires-en espaces vectoriels ou en programmation dans un langage comme le C ;

– Avoir au moins 17 ans.

Si vous ne voulez pas participer à cette étude, vous pouvez juste ignorer ce courriel.

Cette recherche a obtenu une certification éthique du Comité d'éthique de la recherche avec les êtres humains (CER-TELUQ) le 12/04/2022. No de référence : 2022-08.

Vous avez des questions au sujet de cette étude ? Écrivez-nous à ntang.pierre_marie@univ.teluq.ca

Le masculin est utilisé ici juste à titre de simplification. Notre étude ne fait aucune discrimination basée sur le genre.

Cliquez sur le lien pour remplir le questionnaire ou copier et coller l'url qui suit dans votre navigateur https://www.cognitoforms.com/PierreMarieNtang/QuestionnaireGroupe3

RÉFÉRENCES

Algom, D. et Chajut, E. (2019). Reclaiming the stroop effect back from control to input-driven attention and perception. *Frontiers in psychology*, *10*, 1683.

Amiri, H. et Shahbahrami, A. (2020). Simd programming using intel vector extensions. *Journal of Parallel and Distributed Computing*, *135*, 83–100.

Baddeley, A. D. et Lieberman, K. (2017). Spatial working memory. In *Exploring working memory* 206–223. Routledge.

Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick, D. L. et Stokes, R. A. (1968). The illiac iv computer. *IEEE Transactions on computers*, *100*(8), 746–757.

Barredo, A., Cebrian, J. M., Moretó, M., Casas, M. et Valero, M. (2020). Improving predication efficiency through compaction/restoration of simd instructions. Dans *2020 IEEE international symposium on high performance computer architecture (HPCA)*, 717–728. IEEE.

Bedu, L., Tinh, O. et Petrillo, F. (2019). A tertiary systematic literature review on software visualization. Dans *2019 Working Conference on Software Visualization (VISSOFT)*, 33–44. IEEE.

Bertin, J. (1983). *Semiology of graphics*. University of Wisconsin press.

Biermann, A. W. et Feldman, J. A. (1972). On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Transactions on Computers*, *C-21*(June), 592–597. `http://dx.doi.org/10.1109/TC.1972.5009015`

Bjøner, D. (2006). Requirements verification and validation. *Software Engineering 3 : Domains, Requirements, and Software Design*, 503–509.

Borgo, R., Micallef, L., Bach, B., McGee, F. et Lee, B. (2018). Information visualization evaluation using crowdsourcing. Dans *Computer Graphics Forum*, volume 37, 573–595. Wiley Online Library.

Brooks Jr, F. P. (1996). The computer scientist as toolsmith ii. *Communications of the ACM*, *39*(3), 61–68.

Carnap, R. (1997). Signification et nécessité, trad. f. rivenc et p. de rouilhan.

Cebrian, J. M., Natvig, L. et Jahre, M. (2020). Scalability analysis of AVX-512 extensions. *Journal of Supercomputing*, *76*(3), 2082–2097. `http://dx.doi.org/10.1007/s11227-019-02840-7`

Chambadal, L. et Ovaert, J. (1966). *Cours de mathématiques : Notions fondamentales d'algèbre et d'analyse*. Numéro v. 1. Gauthier-Villars. Récupéré de `https://books.google.ca/books?id=xE-EnQAACAAJ`

Chen, M., Feixas, M., Viola, I., Bardera, A., Shen, H.-W. et Sbert, M. (2016). *Information theory tools for visualization*. AK Peters/CRC Press.

Chi, E. H.-h. (2000). A taxonomy of visualization techniques using the data state reference model. Dans *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, 69–75. IEEE.

Chotisarn, N., Merino, L., Zheng, X., Lonapalawong, S., Zhang, T., Xu, M. et Chen, W. (2020). A systematic literature review of modern software visualization. *Journal of Visualization*, *23*(4), 539–558.

Christensen, H. B. (1986). *La statistique : démarche pédagogique programmée*. Chicoutimi, Québec : G. Morin.

Cicchello, O. et Kremer, S. C. (2004). Inducing grammars from sparse data sets : A survey of algorithms and results. *Journal of Machine Learning Research*, *4*(4), 603–632. http://dx.doi.org/10.1162/153244304773936063

Clang (2020). Clang Language Extensions. Récupéré le 27 may 2021 de https://clang.llvm.org/docs/LanguageExtensions.html#vectors-and-extended-vectors

Correll, M. (2022). Are we making progress in visualization research ? *arXiv preprint arXiv :2208.11810*.

Dasgupta, S., Park, D., Kasampalis, T., Adve, V. S. et Roşu, G. (2019). A complete formal semantics of x86-64 user-level instruction set architecture. Dans *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1133–1148. ACM.

Diehl, S. (2007). *Software visualization : visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media.

Dirty hands coding (2019). utf8lut : Vectorized UTF-8 converter. Decoding UTF-8. https://dirtyhandscoding.github.io/posts/utf8lut-vectorized-utf-8-converter-introduction.html.

Dupont, P., Lambeau, B., Damas, C. et Van Lamsweerde, A. (2008). The QSM algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, *22*(1-2), 77–115. http://dx.doi.org/10.1080/08839510701853200

Edelsohn, D., Gellerich, W., Hagog, M., Naishlos, D., Namolaru, M., Pasch, E., Penner, H., Weigand, U. et Zaks, A. (2005). Contributions to the gnu compiler collection. *IBM Systems Journal*, *44*(2), 259–278. http://dx.doi.org/10.1147/sj.442.0259

Engelhardt, Y. et Richards, C. (2020). *The DNA Framework of Visualization*, volume 12169 LNAI. Springer International Publishing. http://dx.doi.org/10.1007/978-3-030-54249-8_51. Récupéré de http://dx.doi.org/10.1007/978-3-030-54249-8{_}51

Ernst, M. D., Cockrell, J., Griswold, W. G. et Notkin, D. (2001). Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, *27*(2), 99–123. http://dx.doi.org/10.1109/32.908957

Erwig, M., Smeltzer, K. et Wang, X. (2017). What is a visual language ? *Journal of Visual Languages & Computing*, *38*, 9–17.

Estérie, P., Falcou, J., Gaunard, M. et Lapresté, J.-T. (2014). Boost. simd : generic programming for portable simdization. Dans *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing*, 1–8.

Ewart, T., Delalondre, F. et Schürmann, F. (2014). Cyme : A library maximizing simd computation on user-defined containers. Dans *Proceedings of the 29th International Conference on Supercomputing - Volume 8488*, ISC 2014, 440–449., New York, NY, USA. Springer-Verlag New York, Inc. http://dx.doi.org/10.1007/978-3-319-07518-1_29

Farghally, M. F., Koh, K. H., Shahin, H. et Shaffer, C. A. (2017). Evaluating the effectiveness of algorithm analysis visualizations. Dans *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, 201–206., New York, NY, USA. ACM. http://dx.doi.org/10.1145/3017680.3017698

Futrelle, R. P. (1999). Ambiguity in visual language theory and its role in diagram

parsing. Dans *Proceedings 1999 IEEE Symposium on Visual Languages*, 172–175. IEEE.

Godbolt (2020). Compiler explorer. `https://godbolt.org/`.

Gross, M. (2016). Neat SIMD : Elegant vectorization in C++ by using specialized templates. Dans *High Performance Computing & Simulation (HPCS), 2016 International Conference on*, 848–857. IEEE.

Hakone, A., Harrison, L., Ottley, A., Winters, N., Gutheil, C., Han, P. K. et Chang, R. (2016). Proact : iterative design of a patient-centered visualization for effective prostate cancer health risk communication. *IEEE transactions on visualization and computer graphics*, *23*(1), 601–610.

Hennessy, J. L. et Patterson, D. A. (2011). *Computer architecture : a quantitative approach*. Elsevier.

Intel (2011). Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes. *System*, *3*(253665). `http://dx.doi.org/10.1109/MAHC.2010.22`

Intel (2018). Intel intrinsics guide. `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#text=_mm256_maskz_unpacklo_epi32`.

Intel (2018). Intrinsics Guide. Récupéré le 26 may 2021 de `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`

Isaacs, K. E., Bremer, P.-T., Jusufi, I., Gamblin, T., Bhatele, A., Schulz, M. et Hamann, B. (2014). Combing the communication hairball : Visualizing parallel execution traces using logical time. *IEEE transactions on visualization and computer graphics*, *20*(12), 2349–2358.

ITU — International Telecommunication Union (2023). Facts and Figures 2022. https://www.itu.int/hub/publication/d-ind-ict_mdd-2022/.

Karpiński, P. et McDonald, J. (2017). A high-performance portable abstract interface for explicit simd vectorization. Dans *Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores*, 21–28.

Kim, M. C., Zhu, Y. et Chen, C. (2016). How are they different ? a quantitative domain comparison of information visualization and data visualization (2000–2014). *Scientometrics*, *107*(1), 123–165.

Kretz, M. et Lindenstruth, V. (2012). Vc : A C++ library for explicit vectorization. *Software : Practice and Experience*, *42*(11), 1409–1430.

Krzikalla, O. et Zitzlsberger, G. (2016). Code Vectorization Using Intel Array Notation. Dans *Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing*, WPMVP '16, 6 :1–6 :8., New York, NY, USA. ACM. http://dx.doi.org/10.1145/2870650.2870655

Kusswurm, D. (2018). *Modern X86 Assembly Language Programming : Covers X86 64-bit, AVX, AVX2, and AVX-512*. Apress.

Kwon, T. et Su, Z. (2011). Modeling high-level behavior patterns for precise similarity analysis of software. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 1134–1139. http://dx.doi.org/10.1109/ICDM.2011.104

Lakatos, I. (1970). History of science and its rational reconstructions. Dans *PSA : Proceedings of the biennial meeting of the philosophy of science association*, volume 1970, 91–136. D. Reidel Publishing.

Lakatos, I. (1976). Falsification and the methodology of scientific research programmes. In *Can theories be refuted ?* 205–259. Springer.

Lakatos, I. (1980). *The methodology of scientific research programmes : Volume 1 : Philosophical papers*, volume 1. Cambridge university press.

Lakatos, I. et Musgrave, A. (1969). Problems in the philosophy of science. *British Journal for the Philosophy of Science*, *20*(1).

Lakoff, G. et Johnson, M. (2008). *Metaphors we live by*. University of Chicago press.

Lee, J., Petrogalli, F., Hunter, G. et Sato, M. (2017). Extending OpenMP SIMD Support for Target Specific Code and Application to ARM SVE. Dans *International Workshop on OpenMP*, 62–74. Springer.

Leissa, R., Haffner, I. et Hack, S. (2014). Sierra : A SIMD Extension for C++. Dans *Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing*, WPMVP '14, 17–24., New York, NY, USA. ACM. http://dx.doi.org/10.1145/2568058.2568062

Lemire, D., Piotte, J., Ntang, P. M. et Bjornson, Z. (2019). Example failure - AVX512F/VL intrinsic support. Récupéré le 28 may 2021 de https://github.com/piotte13/SIMD-Visualiser/issues/33

Levkowitz, H. (1997). *Color theory and modeling for computer graphics, visualization, and multimedia applications*, volume 402. Springer Science & Business Media.

Li, B., Mooring, J., Blanchard, S., Johri, A., Leko, M. et Cameron, K. W. (2017). Seemore. *J. Parallel Distrib. Comput.*, *105*(C), 183–199. http://dx.doi.org/10.1016/j.jpdc.2017.01.017

LLVM (2018). The llvm compiler infrastructure. https://llvm.org/.

Lorensen, B. (2004). On the death of visualization. *Proceedings - Fall 2004 Workshop Visualization Research Challenges*, *1*, 5.

Lorenzoli, D., Mariani, L. et Pezzè, M. (2008). Automatic generation of software behavioral models. *Proceedings - International Conference on Software Engineering*, 501–510. http://dx.doi.org/10.1145/1368088.1368157

Luck, S. J. et Hollingworth, A. (2008). *Visual memory*. OUP USA.

MacDonald, L. W. (1999). Using color effectively in computer graphics. *IEEE Computer Graphics and Applications*, *19*(4), 20–35. http://dx.doi.org/10.1109/38.773961

Manovich, L. (2011). What is visualization ? *DIGAREC Series*, (6), 116–156.

Marriott, K. et Meyer, B. (1998). *Visual language theory*. Springer Science & Business Media.

Mattila, A.-L., Ihantola, P., Kilamo, T., Luoto, A., Nurminen, M. et Väätäjä, H. (2016). Software visualization today : Systematic literature review. Dans *Proceedings of the 20th International Academic Mindtrek Conference*, 262–271.

Mazza, R. (2009). *Introduction to information visualization*. Springer Science & Business Media.

Merino, L., Ghafari, M., Anslow, C. et Nierstrasz, O. (2018). A systematic literature review of software visualization evaluation. *Journal of systems and software*, *144*, 165–180.

Meyer, M. (1979). Découverte et justification en science : kantisme, néo-positivisme et problématologie.

Mitra, G., Johnston, B., Rendell, A. P., McCreath, E. et Zhou, J. (2013). Use of simd vector operations to accelerate application code performance on low-powered arm and intel platforms. Dans *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 1107–1116. IEEE.

Muła, W. et Lemire, D. (2018). Faster base64 encoding and decoding using AVX2 instructions. *ACM Trans. Web*, *12*(3). http://dx.doi.org/10.1145/3132709

Munzner, T. (2008). Process and pitfalls in writing information visualization research papers. *Lecture Notes in Computer Science*, *4950 LNCS*, 134–153. http://dx.doi.org/10.1007/978-3-540-70956-5_6

Munzner, T. (2009). A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 921–928. http://dx.doi.org/10.1109/TVCG.2009.111

Munzner, T. (2014). *Visualization Analysis and Design*. A K Peters/CRC Press. http://dx.doi.org/10.1201/b17511

Murphy, S. J. (2009). The power of visual learning in secondary mathematics education. *Research into practice mathematics*, 1–8.

Myers, B. A. (1986). Visual programming, programming by example, and program visualization : A taxonomy. Dans *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, 59–66., New York, NY, USA. ACM. http://dx.doi.org/10.1145/22627.22349

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, *1*(1), 97–123. http://dx.doi.org/10.1016/S1045-926X(05)80036-9

Nagel, H. R. (2006). Scientific visualization versus information visualization. Dans *Workshop on state-of-the-art in scientific and parallel computing, Sweden*, 8–9. Citeseer.

Nathan, M. J., Koedinger, K. R. et Alibali, M. W. (2001). Expert blind spot : When content knowledge eclipses pedagogical content knowledge. Dans *Proceedings of the third international conference on cognitive science*, 644–648. Beijing : University of Science and Technology of China Press.

Norman, D. et Draper, S. (1986). New perspectives on human-computer interaction.

Ntang, P.-M. et Lemire, D. (2021). Simdgiraffe : Visualizing simd functions. Dans *VISIGRAPP (3 : IVAPP)*, 147–154.

Nualart-Vilaplana, J., Pérez-Montoro, M. et Whitelaw, M. (2014). How we draw texts : A review of approaches to text visualization and exploration. *El profesional de la información*, *23*(3).

Olivier, M. (2018). *Statistiques appliquées avec introduction au logiciel R*. Ellipses Marketing.

Oppermann, M. et Munzner, T. (2020). Data-first visualization design studies. Dans *2020 IEEE Workshop on Evaluation and Beyond-Methodological Approaches to Visualization (BELIV)*, 74–80. IEEE.

Palmer, S. et Rock, I. (1994). Rethinking perceptual organization : The role of uniform connectedness. *Psychonomic bulletin & review*, *1*(1), 29–55.

Palmeri, T. J. et Tarr, M. (2008). Visual object perception and long-term memory. *Visual memory*, 163–207.

Papenhausen, E., Mueller, K., Langston, M. H., Meister, B. et Lethin, R. (2016). An interactive visual tool for code optimization and parallelization based on the polyhedral model. Dans *Parallel Processing Workshops (ICPPW), 2016 45th International Conference on*, 309–318. IEEE.

Peirce, S. C. et Bucher, J. (1955). *Philosophical writings of Peirce*. http://dx.doi.org/10.5840/teachphil200629223

Petre, M. et Blackwell, A. (1998). Cognitive questions in software visualization. *Software visualization :*, 1–23.

Pohl, A., Cosenza, B., Mesa, M. A., Chi, C. C. et Juurlink, B. (2016). An evaluation of current simd programming models for c++. Dans *Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing*, WPMVP '16, 3 :1–3 :8., New York, NY, USA. ACM. http://dx.doi.org/10.1145/2870650.2870653

Popper, K. (2005). *The logic of scientific discovery*. Routledge.

Price, B., Baeker, R. et Small, I. (1998). An introduction to software visualization. In J. A. DOMINGUE (dir.), *Software visualization : Programming as a multimedia experience* chapitre I, 3–27. MIT press.

Price, B. A., Baecker, R. M. et Small, I. S. (1993). A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, *4*(3), 211–266.

Purchase, H. C., Andrienko, N., Jankun-Kelly, T. J. et Ward, M. (2008). Theoretical foundations of information visualization. *Lecture Notes in Computer Science*, *4950 LNCS*, 46–64. http://dx.doi.org/10.1007/978-3-540-70956-5_3

Quinn, M. (2003). *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill.

Reichenbach, H. (1938). Experience and prediction : An analysis of the foundations and the structure of knowledge.

Robert, S. (2009). Logique de la découverte et naturalisation de la connaissance : L'épistémologie historique d'imre lakatos. *Presses de l'Université Laval, Québec*.

Robertson, G., Fernandez, R., Fisher, D., Lee, B. et Stasko, J. (2008). Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1325–1332. http://dx.doi.org/10.1109/TVCG.2008.125

Schmidt, B., González-Domínguez, J., Hundt, C. et Schlarb, M. (2018). *Parallel Programming Concepts and Practice*. Morgan Kaufmann.

Schurgin, M. W. (2018). Visual memory, the long and the short of it : A review of visual working memory and long-term memory. *Attention, Perception, & Psychophysics*, *80*(5), 1035–1056.

Sedig, K. et Parsons, P. (2016). Design of visualizations for human-information interaction : A pattern-based framework. *Synthesis Lectures on Visualization*, *4*(1), 1–185.

Sedlmair, M., Meyer, M. et Munzner, T. (2012). Design study methodology : Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2431–2440. http://dx.doi.org/10.1109/TVCG.2012.213

Sensalire, M., Ogao, P. et Telea, A. (2009). Evaluation of software visualization tools : Lessons learned. Dans *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 19–26. IEEE.

Seriai, A., Benomar, O., Cerat, B. et Sahraoui, H. (2014). Validation of software visualization tools : A systematic mapping study. Dans *2014 Second IEEE Working Conference on Software Visualization*, 60–69. IEEE.

Spence, R. (2014). *Information Visualization : An Introduction*. Springer.

St. Amant, R. (1998). Planning and user interface affordances. Dans *Proceedings of the 4th international conference on Intelligent user interfaces*, 135–142.

Standing, L. (1973). Learning 10000 pictures. *The Quarterly journal of experimental psychology*, *25*(2), 207–222.

Steigerwald, B. et Agrawal, A. (2011). Developing Green Software. *Intel White Paper*, 1–11.

Stojanov, A., Toskov, I., Rompf, T. et Püschel, M. (2018). Simd intrinsics on managed language runtimes. Dans *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2–15.

Strobelt, H., Oelke, D., Kwon, B. C., Schreck, T. et Pfister, H. (2015). Guidelines for effective usage of text highlighting techniques. *IEEE transactions on visualization and computer graphics*, *22*(1), 489–498.

Stupachenko, E. V. (2015). Programming using AVX2. Permutations. Récupéré le 27 may 2021 de `https://software.intel.com/content/www/us/en/develop/blogs/programming-using-avx2-permutations.html`

Tardif, A. (2015). Contexte de découverte et contexte de justification : une aporie dans l'empirisme logique ? *Revue Phares*, *15*.

Van Hoey, J. (2019). *Beginning X64 Assembly Programming : From Novice to AVX Professional*. Apress.

van Wijk, J. (2006). The Value of Visualization. *VIS 05. IEEE Visualization, 2005.*, 79–86. http://dx.doi.org/10.1109/visual.2005.1532781

VPUNPCK (2018). Intel® avx-512 instructions. https://software.intel.com/en-us/blogs/2015/01/13/programming-using-avx2-permutations.

Wang, H., Andrade, H., Gedik, B. et Wu, K.-L. (2009). Auto-vectorization through code generation for stream processing applications. Dans *Proceedings of the 23rd international conference on Supercomputing*, 495–496.

Wang, H., Wu, P., Tanase, I. G., Serrano, M. J. et Moreira, J. E. (2014). Simple, portable and fast simd intrinsic programming : Generic simd library. Dans *Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing*, WPMVP '14, 9–16., New York, NY, USA. ACM. http://dx.doi.org/10.1145/2568058.2568059

Wertheimer, M. (1938). *Gestalt theory.* Kegan Paul, Trench, Trubner & Company.

Wertheimer, M. (2012). *On perceived motion and figural organization.* MIT Press.

Wilhelm, A., Savu, V., Amadasun, E., Gerndt, M. et Schuele, T. (2016). A visualization framework for parallelization. Dans *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, 81–85. IEEE.

Zhang, K. (2010). *Visual languages and applications.* Springer Science & Business Media.