UNIVERSITÉ DU QUÉBEC À MONTRÉAL

EXTRACTION ET GESTION DES CONNAISSANCES DES ALGORITHMES D'ORDONNANCEMENT

THÈSE
PRÉSENTÉE
DU DOCTORAT EN INFORMATIQUE

PAR
MARTIN DUBOIS

AOÛT 2015

REMERCIEMENTS

Je remercie toutes les personnes qui ont contribué et ont soutenu de près ou de loin la réalisation de ce travail.

Je tiens particulièrement à remercier mon directeur de recherche, le professeur Mounir Boukadoum, pour son encadrement et ses suggestions lors de la réalisation de ce projet.

Je remercie mes parents et mon frère, Mathieu Dubois, pour leur soutien dans ce projet de recherche.

Je dédie cette thèse à ma mère qui nous a quittés trop tôt et envers qui je suis très reconnaissant.

ABSTRACT

Modern technology allows for increasingly dense SoCs with more and more embedded processors, leading to greater software flexibility for application design and implementation. Systems must be able to use the capabilities of chips for execution of various application tasks, which poses the problem of task scheduling. Research in this area has produced a tremendous amount of diversified knowledge. This work addresses the extraction and efficient representation of this knowledge for designers who use SoC and creators of scheduling algorithms.

We propose to use formal concept analysis and data-mining techniques to extract knowledge about scheduling algorithms from a variety of sources such as bench tests and intrinsic characteristics of algorithms. Our research shows how to retrieve performance information for a variety of application contexts, allowing the efficient selection of the best scheduling algorithm for a user application. The information is extracted without recourse to statistical tool, thus avoiding the drawing of general conclusions that may be inappropriate for the specific problem of the user. Knowledge representation is done on several levels and is based on software patterns and languages. Everything is integrated in a platform for knowledge management that can be used to design algorithms and applications more quickly and efficiently.

The formal concept analysis unlocks the capacity of the presented methodology to provide a structured knowledge representation about task scheduling. As a first step, the raw knowledge is transformed into association rules between application attributes and performance metrics for task scheduling. This allows not only the formal manipulation of scheduling knowledge, but also increasing the knowledge of the designer about specific scheduling cases.

A method is also proposed for the visualisation of the obtained knowledge. We show how to automate the knowledge extraction and make a representation in a pattern language. We present a platform for documentation and ways to produce patterns and combine them to help the software architect in the use of scheduling algorithms. In addition, we present different perspectives that combine patterns, association rules and data mining. Finally, a new class of scheduling algorithms is presented, based on mining of association rules and offers improved algorithms.

Our approach helps designers develop their applications and experts to improve and create scheduling methods. Moreover, it can be used to integrate documentation tools in the design process.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABREVIATIONS

MPSoC	Embedded Multiprocessor SoC
Soc	Système sur puce
DSP	Digital Signal Processing
VHDL	High-Level Synthesis Design
CSP	Communicating Sequential Processes
E-sys	Embedded Systems
Cossap	Communication System Simulation and Application Processor
MCSE	Méthodologie de conception des systèmes électroniques
SDL	Specification and Description Language
UML	Unified Modeling Language
Corba	Common Object Request Broker Architecture
FPGA	Field-Programmable Gate Array
FFT	Fast Fourier Transform
GAD	Graphe dirigé acyclique

RÉSUMÉ

Mots clefs : Extraction, analyse formelle, connaissance, multi-cœur, patrons, grammaire, forage, ordonnancement.

La technologie moderne permet d'avoir des systèmes sur puce de plus en plus denses et comportant de plus en plus de processeurs, menant ainsi à une grande flexibilité logicielle pour la conception et la réalisation d'applications. Les systèmes doivent pouvoir utiliser les capacités des puces pour l'exécution de tâches applicatives diverses, ce qui pose le problème de l'ordonnancement des tâches. La recherche dans ce domaine a produit une quantité phénoménale et diversifiée de connaissances. Notre travail porte sur l'extraction et la représentation efficace des connaissances pour les concepteurs qui utilisent les puces et les créateurs d'algorithmes d'ordonnancement.

Nous proposons d'utiliser l'analyse formelle de concept et les techniques du forage de données pour l'extraction de connaissances sur les algorithmes d'ordonnancement, en partant de diverses sources telles les bancs d'essai et les caractéristiques intrinsèques des algorithmes. Notre recherche montre comment extraire efficacement les informations de performance dans un ensemble d'applications, permettant ainsi la sélection efficace du meilleur algorithme d'ordonnancement pour l'application de l'usager. L'extraction de l'information se fait sans outil statistique, évitant ainsi de tirer des conclusions générales qui peuvent être inappropriées pour le problème spécifique de l'usager. La représentation des connaissances est faite sur plusieurs niveaux et se base sur les patrons logiciels et les langages-patrons. Le tout est intégré dans une plateforme de connaissance qui peut être utilisée pour concevoir plus rapidement et efficacement de nouveaux algorithmes d'ordonnancement et des applications.

L'usage de l'analyse formelle de concepts permet à la méthodologie présentée d'offrir une représentation structurée des connaissances sur l'ordonnancement de tâches. Dans un premier temps, les connaissances brutes sont transformées en règles d'associations entre les attributs applicatifs et des métriques de performance pour l'ordonnancement des tâches. Ceci permet non seulement la manipulation formelle du savoir obtenu, mais aussi d'accroitre la connaissance du concepteur au sujet d'ordonnancement précis.

Une méthode pour cartographier des connaissances obtenues est proposée pour une visualisation rapide. Nous montrons aussi comment automatiser l'extraction et l'exploitation du savoir à l'aide d'une représentation en langage-patron. En particulier, nous présentons une plateforme de documentation et des façons de produire des patrons et de les regrouper pour aider l'architecte logiciel dans l'utilisation des algorithmes d'ordonnancement. En plus, nous exposons différentes perspectives qui combinent les patrons, les règles associatives et l'extraction de données. Finalement, une nouvelle classe d'algorithmes d'ordonnancement est présentée, basée sur le forage des règles d'association pour proposer des algorithmes à la performance améliorée.

Notre approche aide les concepteurs à développer leurs applications et les experts à améliorer et créer des méthodes d'ordonnancement. De plus, elle peut servir à intégrer des outils documentaires dans le cycle de design.

CHAPITRE

INTRODUCTION

Les systèmes électroniques et informatiques d'aujourd'hui sont de plus en plus complexes et doivent répondre à des critères de performance de plus en plus élevés. Des exemples typiques sont les cartes graphiques pour les applications vidéo, les ordinateurs de poche de plus en plus intelligents, les circuits d'accès à l'Internet de plus en plus rapides, les centres multimédias et les mécanismes d'annulation d'écho dans les circuits de communication. Au début, le matériel des systèmes reposait sur des portes logiques. Ensuite, ont été ajoutés les microprocesseurs à cœur simple, puis ceux à cœur double. Aujourd'hui, on utilise des processeurs à cœur quadruples, sextuples, octuples, et les puces renfermant 64 processeurs et plus ne sont plus un rêve [1]. En fait, on peut séparer les ordinateurs aujourd'hui selon le nombre de processeurs qu'ils utilisent. On a ainsi ceux relativement simples d'un à deux processeurs, ceux à multiprocesseurs comprenant de trois à une centaine de processeurs et ceux à plusieurs centaines de processeurs comme les produits en développement chez la compagnie Tilera [1]. Il est également question d'employer des puces avec 1000 processeurs et plus dans un futur rapproché [2], [3]. Cela est possible grâce à l'évolution continue des processus de fabrication qui permet de réduire régulièrement les dimensions physiques des puces et d'augmenter ainsi leur densité. Ces avancées technologiques permettent de réaliser de nombreuses architectures matérielles et logicielles, adaptées à divers domaines d'application. Par exemple, la compagnie Octasic [4] utilise 15 DSPs sur une puce pour les applications multimédias évolutives; Nvidia propose le Telsa qui possède 240 processeurs à flux de données, répartis en 10 puces de 24 unités chacune, pour créer un super ordinateur graphique [5], et Tilera propose une puce avec 64 processeurs de 64 bits pour des applications générales [1].

Le secteur d'aide à la conception par ordinateur suit de près les avancées précédentes et est soumis à la pression de développer des outils de plus en plus complexes et performants pour

la réalisation des applications. Dans l'aide à la conception, plusieurs tendances ont vu le jour. Dans les années 1980, les outils disponibles visaient directement l'utilisation de portes logiques pour réaliser des circuits portes logiques ; on cherchait alors des moyens d'automatiser le placement des portes. Dans les années 1990, on parlait d'avoir des langages de représentation matérielle au niveau du transfert de registres (RTL en anglais) [6]. Cela a donné lieu à la création des langages VHDL et Verilog [6] qui sont devenus la norme pour la synthèse matérielle. Vers les années 2000, des spécialistes ont introduit le concept de synthèse fonctionnelle. Celui-ci est modélisé par une méthodologie à plusieurs niveaux qui inclut RTL et portes logiques, auxquels on ajoute des couches supérieures plus abstraites. Les niveaux supérieurs tentent de modéliser autant les fonctions matérielles que logicielles. Le premier niveau supérieur à RTL est le niveau transactionnel. À ce niveau, on retrouve par exemple des modèles de computation et des langages tels CSP, SystemC, E-sys et Cossap [7]- [8]. Un niveau encore plus haut est le niveau *messages* avec, par exemple, des langages de modélisation comme MCSE, SDL et UML [9]. Pour finir, on identifie des technologies orientées composantes telles Corba [8].

L'introduction des systèmes faits de logiciel et de matériel amena aussi l'avènement du codesign et de la cosimulation. Ceux-ci permettent de partitionner une application en composantes logicielles et matérielles. L'approche cherche à bénéficier de la flexibilité du logiciel et des performances du matériel. Les outils disponibles, comme Seamless de Cadence, permettent de simuler au niveau RTL de bas niveau plusieurs types de processeurs et aussi du matériel modélisé en VHDL. Le concepteur peut ainsi simuler l'interaction entre les parties logicielles et matérielles. L'outil de Seamless peut traiter quelques processeurs. On retrouve à chaque niveau des outils d'abstraction spécialisés, capables de prendre une spécification à ce niveau et d'amener celle-ci vers un niveau plus bas.

On utilise à haut niveau le langage C ou C++ et les outils tentent de convertir une description en une réalisation sur puce programmable automatiquement; c'est ce que fait Catapult-C [9] de Mentor Graphics par exemple, qui opère sur un programme rédigé en C. L'usager fournit un code C qui représente son système et l'outil le convertit en VHDL, avec sa partie

logicielle. On utilise des bibliothèques en C pour représenter le système. Le système généré est généralement appelé un système sur puce (SoC en anglais) [9].

Donc, d'un côté, on a les applications et, de l'autre, des systèmes avec des modules et des interconnexions. Par ailleurs, le concepteur doit être capable de décrire une application en un système sans pour autant avoir une idée préconçue de son implémentation. Une fois cette étape franchie, le concepteur se retrouve devant trois approches de réalisation possibles qu'il doit être en mesure d'évaluer.

Dans le premier cas, on réutilise des modules existants. Les outils d'aide à la conception comprennent des bibliothèques de modules et des plateformes que l'on peut réutiliser au niveau matériel. Du côté logiciel, on cherche à réutiliser des parties de code source déjà disponibles. On regroupe les parties sous forme d'artéfacts et on utilise des méthodes de butinage de données, ceci afin de récupérer et réutiliser ce qui existe en vue de réduire le temps de conception. Les architectes de systèmes fournissent et conçoivent les plateformes pour les rendre disponibles sur les marchés de compagnies comme Nvdia, Octasic et Tilera. Les architectes fournissent aux outils de conception des modèles architecturaux qui tiennent compte des modules et de leurs interconnexions. Les plateformes sont caractérisées en matière de débit, de puissance et d'efficacité. Les architectures fournissent une modélisation de plateformes de réalisation et des outils d'analyse de performance. Le concepteur doit définir les paramètres de sa plateforme et avoir une idée des performances de celle-ci pour son application. L'exploration du design demeure aussi une préoccupation avec des choix de plateformes, de modules et de réseaux d'interconnexion.

Dans le deuxième cas, il n'existe pas de matériel. On doit concevoir l'architecture à partir de zéro. Le dernier cas est un hybride des deux premiers cas, par exemple l'utilisation d'un FPGA programmable avec des modules dédiés. Les défis à relever sont la capacité d'exploiter dans un temps de design de plus en plus court une architecture matérielle qui demande parfois un long temps d'apprentissages pour la connaitre en profondeur. De plus, les designs sont de plus en plus gros et complexes, avec une multitude de possibilités d'optimisation et de réalisation. Par exemple, nous pouvons optimiser un design en temps d'exécution, en consommation d'énergie, ou les deux en même temps. Le concepteur doit

être capable d'estimer les capacités de la plateforme cible par rapport aux applications qu'il doit réaliser. Il existe une étape intermédiaire, un lien, entre les fournisseurs de matériel et les applications. Ce lien consiste en la réalisation d'une application pour un matériel ciblé. On doit passer nécessairement par une phase d'exploration. Par conséquent, on dispose d'une application sous forme d'une représentation quelconque, en mesure d'avoir les paramètres des niveaux supérieurs ; en même temps, on doit connaître suffisamment les performances de l'architecture cible choisie. L'exploration architecturale doit avoir impérativement une fonctionnalité séparée de la représentation du système. Elle doit aussi se faire rapidement, être interactive et pouvoir être simulée de façon comportementale à l'aide de modèles architecturaux réalisables.

De façon générale, un réseau de processeurs offre de hautes performances, une flexibilité de conception et une rapidité d'utilisation.–Par contre, il possède une longue période de conception, une consommation d'énergie accrue et une large surface. Il est important de disposer d'outils capables d'exploiter ce genre de puce tout en minimisant le temps de conception. D'un côté, le concepteur possède une description fonctionnelle des modules et il identifie des tâches qu'il souhaite réaliser et, de l'autre côté, il se retrouve devant une architecture cible qui peut comprendre des centaines de processeurs. Dans ce contexte, déterminer comment répartir et ordonnancer les tâches sur un ensemble de processeurs est devenu un défi courant.

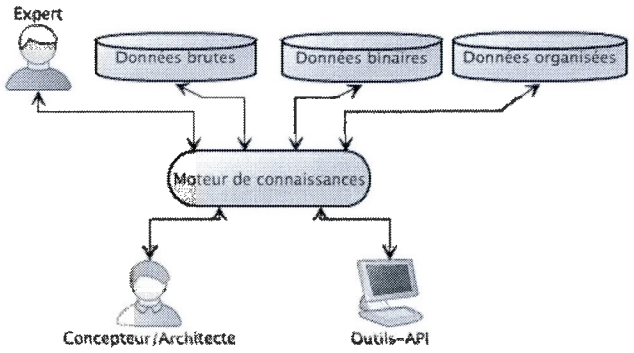
Motivation

Avec les progrès technologiques récents, l'ordonnancement d'une description fonctionnelle et sa caractérisation architecturale dans un système à multiprocesseurs sont devenus des problèmes importants et évoluant en complexité. Une recherche rapide sur le sujet indique des milliers de références qui touchent autant à l'architecture à utiliser qu'à la gestion de paramètres tels la préemption, la redondance des tâches, la décision d'un ordonnancement statique ou dynamique, l'utilisation d'unités de traitement homogènes ou hétérogènes, etc. Ces questions n'ont pas de réponses simples au regard de la diversité de plateformes d'exécution, car les processeurs sur puces peuvent être synchrones ou asynchrones, homogène ou hétérogènes, avoir des mémoires locales distribuées ou externes, des interfaces

de communications diverses, et des fréquences d'opération et tensions d'alimentation programmables. On dispose également de différentes architectures de processeurs telles que celles de Intel, AMD, ARM et Freescale [10]. Dans ce contexte, les algorithmes d'ordonnancement ont à réaliser une tâche importante de planification de l'exécution des modules en tenant compte, entre autres, du temps maximal d'une application, du nombre de processeurs, et de l'efficacité énergétique.

Devant la perspective d'être inondé d'informations diverses concernant les algorithmes et de leurs conditions d'utilisation, il est devenu difficile de choisir le bon algorithme pour ordonnancer une application donnée, et documenter les résultats pour une exploitation ultérieure. La réponse à ces problèmes passe par des manières efficaces de formaliser et d'automatiser l'acquisition des connaissances pertinentes, de les gérer et de les documenter. Nous proposons l'environnement présenté à la figure 0.1 pour répondre à ce besoin. L'environnement manipule différents types de données et un moteur de connaissance qui interagit avec trois types d'usagers : l'expert qui connaît les algorithmes, le concepteur/architecte qui veut implémenter son application, et un/des outil(s) automatique(s) dans le but d'une intégration poussée et cohérente de la gestion du savoir disponible sur l'ordonnancement.

Figure 0.1 : Environnement de connaissances



Dans la suite de cette thèse, nous progresserons au fur et à mesure pour couvrir les principaux éléments d'automatisation et d'analyse formelle des informations pour choisir un algorithme

dans un vaste ensemble de données mal documentées. Ensuite, nous discuterons la manière de garder les connaissances de façon structurée pour une réutilisation plus facile. Dans un premier temps, nous tracerons une version de l'environnement de connaissance avec l'analyse formelle des trois types de clients. Ensuite, nous introduirons les règles d'association comme outil d'extraction du savoir sur un algorithme d'ordonnancement et verrons comment elles peuvent aussi servir à comparer deux algorithmes. Nous étudierons ensuite comment comparer plus que deux algorithmes en les répertoriant et classifiant d'abord à l'aide de patrons et d'un langage de manipulation. Nous terminerons avec une plateforme de connaissance incluant toutes les capacités proposées en mettant l'accent sur la documentation et la réutilisation des décisions prises et des nouvelles possibilités pour les experts, les concepteurs et les autres outils. La thèse vise particulièrement le savoir sur l'ordonnancement logiciel à partir d'un graphe représentant des tâches avec les métriques associées, en se limitant à l'ordonnancement statique, sans préemption, parmi un ensemble d'algorithmes de référence.

Du point de vue applicatif, la thèse répond à un certain nombre de questions : Comment choisir les bons algorithmes d'ordonnancement en tenant compte de la panoplie de paramètres associés à l'application ? Comment les documenter pour la réutilisation et la création de nouveaux algorithmes ? Comment représenter les algorithmes ? Les réponses à ces questions se feront par plusieurs approches novatrices résumées ci-dessous et présentées plus en détail dans différents chapitres.

Techniques principales proposées

- Représentation des connaissances par le concept d'attributs

Une donnée primaire est un attribut tel que défini dans une base de données et dans l'analyse de concepts. Son usage permet de spécifier l'ensemble des caractéristiques d'un algorithme, d'une plateforme, d'une application et des performances sous forme d'attributs. Dans ce travail, les attributs peuvent être groupés dans une liste binaire dont chaque élément spécifie la présence ou absence de l'attribut correspondant. Ceci permettra de passer des données brutes vers des données binaires et organisées

- Forage de données et utilisation de données brutes

Le forage permet d'extraire les informations à partir de la représentation précédente. On peut obtenir, à partir d'un ensemble d'attributs, des connaissances plus spécifiques tout en recoupant l'application, les performances et les algorithmes. L'information recherchée est sous forme des règles d'association. L'algorithme fp-growth [11] est utilisé dans ce travail.

La fouille de données crée un champ d'exploration. Cette approche permet d'ouvrir une nouvelle approche d'exploration automatique. L'usage des données brutes d'une application permet, à l'opposé de ce qui a été communiqué dans la littérature, de ne pas avoir de préconceptions statiques sur un ensemble; ceci permet à l'usager un plus grand éventail d'analyse avec la fouille de données, comme celui possible avec les patrons de conception. Cette proposition aborde la notion d'extraction de connaissance pour un algorithme.

- Exploration et représentation des connaissances

Cette technique est utilisée pour comparer plus d'un algorithme. Une exploration et une représentation sont effectuées à partir des données extraites du forage de données. La cartographie algorithmique est rendue possible ainsi que l'usage de langages-patrons (définis plus loin) pour accroître les connaissances sur l'ordonnancement de tâches spécifiques. Cette technique montre une nouvelle façon de représenter un design, elle permet d'avoir une vue d'ensemble plus grande des mises en œuvre possibles, et cela, par une approche essentiellement visuelle. Les différents paramètres de graphe sont utilisés pour extraire un patron. L'utilisation des concepts de patron et d'antipatron de conception donne une représentation efficace. En fait, les concepts de patron de conception et de langages-patrons procurent des phases d'exploration de solutions et des analyses plus fines.

- Choix du bon algorithme

Le modèle de développement du système Vee [12] peut être utilisé par un usager afin de déterminer un ensemble d'algorithmes pour son application. L'emploi de méthodes tirées du génie logiciel et des systèmes de développement logiciel permet une interaction plus grande entre l'usager et le processus d'optimisation. Il permet au concepteur de système d'avoir une

méthode qui permet, du début à la fin du processus de développement, d'avoir entre autres les documents, les manuels d'usager, le logiciel et pouvoir répondre aux questions : Comment ?, Pourquoi ? Et Qui ? Concernant l'architecture choisie. Par exemple, pour avoir choisi ce type de composante, pour avoir fait cette classe logicielle et par qui. Le modèle rassemble l'ensemble des connaissances sur un projet de développement. La méthode peut être étendue et vue en trois dimensions : Vee tridimensionnelle.

L'approche Vee tridimensionnelle donne l'exploration et l'allocation de tâches; cela favorise l'utilisation de méthodes existantes, comme l'ordonnancement par liste, sans avoir à effectuer un ordonnancement variable selon l'algorithme utilisé pour le design considéré, ce qui revient à utiliser un algorithme dans sa plage d'opération la plus efficace.

- Documentation

À partir des informations extraites et des patrons, l'utilisation des techniques de diagramme de Hasse et des patrons permet d'offrir des outils de documentation efficace pour garder une trace des décisions et options disponibles. Ainsi, le pourquoi et le comment des décisions pourront être documentés.

Contributions

Dans cette recherche, une plateforme de connaissance sur l'ordonnancement des tâches est réalisée et présentée. Ce nouvel outil permet une représentation efficace et structurée des algorithmes d'ordonnancement. Il permet l'usage de langage-patron et de méthode de visualisation pour fournir des connaissances structurées à des usagers non experts, des experts ou des outils. Il permet aussi de documenter les connaissances apprises et de savoir pourquoi on choisit un algorithme ou un autre. Finalement, il élargit les possibilités concernant la conception d'un algorithme particulier et l'analyse du domaine de l'ordonnancement. Tout ceci permet d'accroître et de gérer la connaissance sur l'ordonnancement de tâches en y contribuant avec des nouvelles approches et techniques. À cet égard, la plateforme étudiée possède plusieurs traits innovateurs :

- Une représentation des connaissances homogènes

L'usage et l'adaptation formelle des attributs dans la représentation des connaissances permettent d'obtenir une homogénéisation des divers savoirs et d'aller plus loin dans l'extraction et la compréhension des mécanismes dans l'ordonnancement de tâches.

- Cartographie algorithmique sur l'ordonnancement pour une vue élargie

Une nouvelle vue d'ensemble d'un groupe d'algorithmes permet d'ajouter des connaissances par rapport à l'analyse statistique et d'établir facilement des comparaisons. Ceci apporte aussi une compréhension plus ciblée des algorithmes.

- Adaptation des patrons logiciels du génie logiciel pour une meilleure compréhension

Les patrons logiciels dans l'ordonnancement permettent une technique de représentation plus efficace des connaissances. Ce formalisme aide à la compréhension des données. L'adaptation de celui-ci dans notre contexte permet de tirer le maximum de l'expérience des algorithmes à partir d'une base de données. Ce concept, jumelé à l'analyse formelle de concept, procure un avantage considérable pour automatiser l'ordonnancement des tâches.

- Adaptation du langage-patron pour une centralisation sur l'usager

L'usage de langages-patrons pour l'ordonnancement permet d'établir une collection hiérarchique des patrons logiciels adaptés dans le cadre de cette recherche. Ceci amène une meilleure compréhension de l'ordonnancement des tâches pour un non-expert. Cette collection peut être utilisée dans un outil d'exploration automatisée ou pour donner des informations ciblées à un utilisateur.

- Création d'un outil interactif de connaissance

La plateforme de savoir n'est pas limitée à un banc d'essai et progresse dans le temps. Celle-ci donne du savoir ciblé selon les besoins d'un expert, d'un novice ou d'un programme automatisé. L'interaction permet une compréhension sur les besoins du client. De plus, ce cadre peut être intégré facilement dans une méthode plus large de développement système comme Vee++, une variante de la méthodologie Vee.

- Création, extraction et adaptation de méthode pour augmenter les connaissances

Nous introduisons un système à deux filtres de connaissance pour obtenir des informations pour la résolution de problèmes ciblés d'ordonnancement. Le premier filtre sélectionne les algorithmes potentiels et le second les classes.

- Représentation des algorithmes de quatre différentes classes sous forme de patrons à partir des données disponibles donnant un autre point de vue sur les connaissances algorithmes.

Organisation de la thèse

Cette thèse contient six chapitres. Le premier chapitre introduit les concepts clés qui seront couverts dans cette recherche. Le chapitre I présente une revue de littérature concernant les algorithmes d'ordonnancement. Le chapitre II introduit le concept de plateforme de connaissances et de modélisation des algorithmes d'ordonnancement. Le chapitre III présente entres autres l'analyse de concept pour l'ordonnancement, les règles delta et la fouille de données. Ensuite, le chapitre IV introduit la notion de cartographie algorithmique, permettant d'avoir une vue d'ensemble d'un ensemble de techniques. Le chapitre V couvre l'usage de langage-patron et explique plus en profondeur les techniques des patrons appliquées à l'ordonnancement des tâches. Le chapitre VI approfondit la notion de deux filtres de connaissance et de la documentation (des décisions prises et les alternatives disponibles), ainsi que la synergie de l'environnement proposé. Le dernier chapitre comporte une conclusion générale quant aux notions développées et présentées.

CHAPITRE I

REVUE DE LITTERATURE SUR LES ALGORITHMES D'ORDONNANCEMENT STATIQUE

Dans ce chapitre, nous présentons différents algorithmes d'ordonnancement et une revue de différents concepts pour la représentation et la manipulation des connaissances. Nous décrivons dans un premier temps un groupe d'algorithmes populaires et leur approche à l'ordonnancement, afin de mettre en relief les différentes informations et connaissances pertinentes. Dans un deuxième temps, nous regardons les différents mécanismes utilisés pour la représentation du savoir dans le but de modéliser et documenter les algorithmes et les décisions prises.

1.1 La problématique des algorithmes d'ordonnancement

Dans plusieurs domaines, nous avons des applications logicielles de télécommunication de pharmaceutique et de l'infonuagique, pour effectuer des tâches courantes. Ces tâches peuvent être séparées en plusieurs tâches atomiques qui peuvent être réalisées en un moment donné.

Nous devons être capables de les segmenter en modules logiciels distribués sur un ensemble de processeurs.

Par exemple, un routeur LTE qui comporte des modules de traitements pour chaque usager doit avoir un délai d'exécution maximal et être mis sur un processeur. Ce choix peut être fait manuellement ou par des outils d'ordonnancement. Cette affectation des modules sur un processeur doit tenir compte de plusieurs facteurs, dont les ressources disponibles et la capacité d'exécution du système, pour rencontrer les performances voulues. Par conséquent, les concepteurs ont recours à des ordonnanceurs. Ceux-ci permettent donc de piloté l'exécution d'une liste de modules logiciels atomiques sur un ensemble de processeurs tout en considérant les critères de performances souhaités.

Il existe deux types d'ordonnanceurs : les statiques et les dynamiques. Le premier type pilote l'exécution pendant le lancement de l'application. Dans le cadre de ce travail, nous concentrons nos efforts sur les méthodes statiques de pilotages de modules avant l'exécution de l'application. Il existe une panoplie de méthodes d'ordonnancement statique dans littérature, le concepteur doit alors choisir et utilisé un ordonnanceur quelconque pour trouver un ordre d'exécution des processus du programme. De plus, il doit être un expert des ordonnanceurs statiques pour savoir celui qui est le mieux pour son application.

Dans cette optique, de quelle manière, devrions-nous représenter les connaissances hétérogènes sur les ordonnanceurs pour aider notre concepteur d'applications à faire son choix selon son application et l'architecture ciblée. Dans un second temps, est-ce que nous pouvons mettre en place une méthodologie, une approche pour choisir automatiquement un ordonnanceur statique pour notre concepteur tout en fournissant de façon effective les raisons de ce choix de conception logiciel.

Dans la conception des systèmes informatiques ou électroniques tels un système téléphonique ou un système d'imagerie médicale, on utilise un modèle pour réaliser ces systèmes. Le modèle vise généralement une architecture technologique cible et répond à une liste de contraintes. Dans les systèmes parallèles, la version simplifiée de Topcuoglu et al. [13] montre que leur programmation comporte des tâches qui doivent être ordonnancées dynamiquement ou statiquement, selon un graphe interactif ou de préséance. Pour définir l'ordonnancement, on représente couramment l'application à l'aide un graphe acyclique dirigé (GAD) G = {M, E}, où M est un ensemble de tâches atomiques et E un ensemble d'arcs entre chaque paire de tâches où chaque arc représente une communication entre les tâches. Par ailleurs, un nœud d'entrée et un de sortie peuvent constituer des pseudo-nœuds. Ceux-ci sont des nœuds ajoutés artificiellement pour n'avoir qu'un seul nœud qui rattache les derniers ou les premiers nœuds de l'application. Par exemple, une application avec 3 nœuds sans sortie peut être reliée artificiellement à un pseudo-nœud de sortie ayant ces trois nœuds comme entrée.

La profondeur du GAD affecte la performance en termes du temps d'exécution. Un arc possède une source, un nœud parent, et sa cible est un nœud enfant. Un nœud qui n'a pas de parent est un nœud racine ou d'entrée. Un nœud qui n'a pas d'enfant est un nœud de sortie. Chaque nœud possède un coût d'exécution. Celui-ci estime le temps que la tâche prend sur une unité d'exécution. L'arc dans l'ensemble E possède aussi un poids qui représente le coût de communication entre deux nœuds. Une unité de traitement peut être un processeur possédant des caractéristiques telles la surface S, le temps d'exécution C, l'efficacité énergétique E, etc. Ces caractéristiques doivent toutes être considérées lors de la réalisation du système. En plus, l'ensemble M peut être exécuté sur une architecture technologique variable. Celle-ci peut être définie par un ensemble Q de P unités de traitement. Q possède trois paramètres principaux : S, V, et P_S. S est la surface, V est la cadence et P_S est la consommation d'énergie. L'architecture peut aussi contenir des caractéristiques additionnelles telles F la fréquence d'opération, T la tension d'alimentation et R la topologie. Chaque unité de traitement peut comporter des spécifications distinctes. Le concepteur est donc appelé à optimiser un ensemble de paramètres visant un ensemble de contraintes. De plus, des contraintes de robustesse peuvent apparaitre. Kwai et Parhami [14] ont déjà montré que le problème de l'ordonnancement de multiprocesseurs est un problème d'optimisation NP-Complet.

Les approches utilisées sont presque aussi nombreuses que les applications et les cibles architecturales ou technologiques. Tout d'abord, on sépare les approches d'ordonnancement et d'allocation en deux groupes selon l'architecture cible : les plateformes homogènes Ho et hétérogènes He. Un réseau homogène possède un ensemble Q uniforme de processeurs P ayant des caractéristiques de communication variables et d'exécution fixes selon le type d'instruction. Un système hétérogène est un ensemble Q variable, avec P pouvant être des processeurs différents ayant des communications variables et des coûts de communication variables d'un P à l'autre pour un type d'instruction. Une architecture peut avoir des topologies variables pour connecter les P dans l'ensemble Q.

Les topologies peuvent être pleinement connectées ou arbitrairement connectées tel que le montrent les réseaux en étoile, en chordal, en hyper cube et en bus de Saldaña et al. [15]. Selon Ali et Rewini [16], les topologies peuvent être caractérisées par le degré des nœuds, le diamètre, la complexité des liens, la largeur bissectrice et la régularité. L'ordonnancement est également conçu à partir de perspectives architecturales diverses et variées : les systèmes homogènes caractérisent des processeurs identiques tandis que les systèmes hétérogènes utilisent des processeurs différents et peuvent être divisés en groupes de processeurs [17] incompatibles, partiellement incompatibles ou compatibles, en fonction du rapport de coût des différents processeurs. D'autre part, il existe plusieurs méthodes de calcul du coût d'un nœud ou d'un arc [18]. Dans la mesure où elles ont recours à des politiques d'ordonnancement différentes en matière de GAD, les approches peuvent générer des performances distinctes pour un seul et même algorithme.

L'ordonnancement est étudié et mis en œuvre par plusieurs auteurs [19], [20], [21], [22], [23], [24], [25], [18], [26]. L'utilisateur a la charge d'exécuter une application ainsi qu'une architecture et se doit d'optimiser au maximum le processus tout en respectant une ou plusieurs contraintes. Différentes méthodes permettent de planifier l'exécution des nœuds en fonction d'un ensemble de processeurs [19], [22], [27], [26], [18]. Les techniques utilisées se concentrent le plus souvent sur l'optimisation d'une quantité limitée de paramètres tels que la vitesse, le nombre de processeurs ou le conflit de transmission.

On regroupe plusieurs approches algorithmiques pour mettre les tâches sur un réseau homogène de processeurs selon la structure du GAD. Toutes les approches mentionnées dans cette section visent principalement à optimiser le temps maximum d'exécution avec des tâches non préemptives. Certains auteurs [28], [29], [16] proposent des algorithmes d'ordonnancement de G à structure spécifique comme un arbre, un complément de graphe chordal et d'ordre d'intervalle. Hu [28] et Coffman [30] créent des méthodes pour des graphes avec des structures arbitraires, mais avec des tâches unitaires d'exécution.

De leur côté, Adam et al. [31] et Kasahara et Narita [32] considèrent des arbres et des coûts de tâches arbitraires, mais sans communication. D'autres spécialistes [33], [34], [35], [36] ajoutent la possibilité de dupliquer les tâches avec des coûts de communication. Ces trois

groupes ont tenu compte des GADs, des communications et des tâches sous différents aspects sans l'utilisation de la duplication.

Certains auteurs [37], [38], [39] regardent l'aspect des tolérances aux pannes d'un ou plusieurs liens de communication ou de processeurs. Le premier groupe [40], [41], [42], [43] introduit la notion de nombre de processeurs infini. Le deuxième groupe et le troisième groupe ont travaillé sur un nombre fini de processeurs en utilisant deux approchent différentes. Plusieurs concepteurs [31], [44], [45], [46], [41] considèrent le réseau de processeurs pleinement connecté—alors que Sih et al. et [47] ainsi que Mehdiratta et autres [48] considèrent une approche avec des connexions arbitraires. Sakellariou et Zhao [18] utilisent quatre types de GAD : les graphes aléatoires, les FFTs, les graphes Fourche-Join et les graphes Laplace. Kwok et Ahmad [25] comparent 27 algorithmes pour des systèmes homogènes, mais ne tiennent pas compte des algorithmes génétiques et de la recherche guidée. Kwok et Ahmad [49] examinent 15 algorithmes et démontrent que certains algorithmes de certains types sont meilleurs que d'autres selon l'architecture cible pour un G donné.

Les types peuvent être des algorithmes basés sur le chemin critique, vorace ou non. Les types sont groupés principalement en un ensemble d'heuristiques et de méthodologie de conception algorithmique tels que l'ordonnancement par liste et les algorithmes génétiques. Le principal défaut de l'analyse est qu'elle est réalisée sur des moyennes et des cas spécifiques, mais elle introduit la notion de ratio du coût de communication sur le coût d'exécution qu'on nomme CCR d'un DAG. Le CCR et le nombre de tâches du GAD ont un impact sur les résultats. Il existe aussi d'autres méthodes basées sur le recuit simulé [50], les colonies de fourmis [51], la recherche locale [52] et les essaims particulaires [53]. Elles ont comme objectif de minimiser le temps d'exécution d'un ensemble de tâches, comme c'est le cas pour les algorithmes génétiques [54], [55], [56], [57], [58], [59], [60], [61] qui donnent une bonne qualité d'ordonnancement, mais avec un temps d'exécution plus long que les autres méthodes. Plusieurs paramètres doivent être considérés et ceux-ci influencent les résultats. Comme alternative, Harik et Lobo [62] proposent une méthode d'adaptation automatique des paramètres.

Des chercheurs [63], [64], [65] utilisent l'optimisation extrême généralisée. Cette approche permet d'utiliser le concept d'espèce et de considérer beaucoup moins de paramètres que les algorithmes génétiques. Hou et al. [66] optimisent spécifiquement une application de virtualisation d'applications. D'autres encore [67], [68], [69] se préoccupent de la dissipation thermique de la puce. De leur côté, Shroff et autres [70] conçoivent un algorithme pour une architecture maître esclave.

On peut élargir toutefois la caractérisation. Sakellariou et Zhao [18] distinguent trois types de systèmes hétérogènes. Ce point est important dans la modélisation d'un DAG. Le premier type est le système hétérogène consistant : n'importe quelle tâche qui s'exécute sur une machine *i* est plus vite que sur une machine *j*. Le deuxième type possède une consistance partielle, définie par le fait que n'importe quelle tâche qui s'exécute sur une machine *i* est plus vite que sur une machine *j* mais seulement pour la moitié des tâches. Le dernier type est le système hétérogène inconsistant : les tâches ont un temps d'exécution aléatoire et n'ont pas de consistance. Dans tous les cas, on utilisera des poids aléatoires répartis sur une plage de données. Cela affectera les différents résultats et, dans certains cas, détériorera les performances des algorithmes. On définit la sensibilité des algorithmes comme étant la variation des performances de ces derniers versus la façon de pondérer les poids.

Au moins quatre classes d'algorithmes ont été proposées dans le cadre du problème d'ordonnancement : liste, regroupement, duplication et randomisation. Les méthodes relatives à l'ordonnancement de liste comprennent PETS (Exécution de l'ordonnancement de la tâche effective), HEFT (hétérogène à durée totale d'exécution minimale) ainsi que CPOP (chemin critique sur un processeur) [13], [22] et reposent sur les différents niveaux de coordination des tâches et la priorité accordée aux nœuds; chaque nœud est doté d'une priorité et l'algorithme planifie ces derniers en commençant par celui dont la priorité est la plus élevée. La méthode CASS (Système de regroupement et d'ordonnancement) [71] rassemble les nœuds en groupes et tire profit du fait de disposer de plus de processeurs pour réduire la longueur de l'ordonnancement. La méthode concernant l'algorithme de duplication CPFD (Chemin critique à duplication rapide) [72] permet le clonage de nœuds; la multiplicité des nœuds en résultant permet de minimiser les retards de communication. Les méthodes de

randomisation utilisent principalement le recuit simulé ou génétique [73], [74], [75]; leur mise en œuvre est simple et elles peuvent mener à des résultants performants en termes de temps d'exécution.

L'ensemble des techniques énumérées ci-dessus suppose l'utilisation d'architectures précises et se révèle particulièrement efficace pour certaines architectures spécifiques. La valeur finale de leurs solutions dépend par conséquent du graphe et des critères de performances utilisés, de l'architecture ciblée, etc.

1.1.1 Métriques de performance et estimateurs

Pour comparer les algorithmes, on utilise plusieurs métriques. De nombreux outils de mesure de performances ont été proposés pour la comparaison des algorithmes. Les outils de mesure peuvent être séparés en deux groupes. Le premier évalue l'algorithme en lui-même. La durée du programme normalisée (NSL - *Normalized Schedule Length*), le rapport de la durée d'ordonnancement (SLR - *Scheduling Length Ratio*) et le taux d'accélération [49], [22], sont des exemples de ces outils de mesure. NSL définit la durée du système, à savoir le temps le plus long entre la première et la dernière tâche; SLR représente le rapport entre la durée et le chemin critique défini comme le plus long chemin dans un GAD, y compris tous les coûts des arcs et les coûts du nœud de calcul; le taux d'accélération pour une application ou un graphe donné est le rapport du temps d'exécution entre une exécution séquentielle et une exécution parallèle.

Le second groupe d'outils de mesure sert à comparer les algorithmes. Le pourcentage de dégradation moyen (APD – *Average Percentage Degradation*) et le nombre de solutions meilleures (NB - *Number of Best Solutions*) [18] sont des exemples types. APD évalue combien les performances d'un algorithme sont inférieures par rapport à un autre et NB est le nombre de fois où un algorithme affichera une performance supérieure par rapport à un autre.

D'autres facteurs influencent les résultats, tels que le fait de prendre la moyenne : Sakellariou et Zhao [18] montrent que la pondération des poids peut affecter les résultats. Par exemple, on peut prendre la valeur moyenne, la valeur médiane, la pire valeur; l'algorithme donnera

des résultats différents. La conclusion a été de segmenter G en sous-graphes afin de diminuer les effets des variations des performances dues aux méthodes d'estimation des métriques. Plus précisément, la variabilité des résultats est diminuée par rapport aux choix de la méthode de pondération des poids (i.e., valeur moyenne, valeur pire, valeur médiane) par l'algorithme d'ordonnancement. Les auteurs utilisent les métriques précédentes pour comparer leurs approches à d'autres. Chacun choisit un ensemble d'algorithmes et tire une conclusion qui varie selon le cas en utilisant des moyennes et des approximations. Il faut recommencer l'analyse à chaque fois que de nouveaux algorithmes généralistes sont trouvés. Par ailleurs, ces outils de mesure sont efficaces pour l'évaluation globale des performances d'un algorithme, mais ils ne sont pas nécessairement à l'échelle d'une application particulière. Dans ce cas, il peut être plus utile de dériver les connaissances désirées à partir d'informations sur le profil GAD. En particulier, le profil GAD peut contenir sa largeur, sa profondeur et son rapport communication-calcul (CCR - *Communication-to-Computation Ratio*), défini comme le coût de communication total de chaque arc sur le coût de calcul total de chaque nœud. Cette information permet le profilage d'un algorithme et permet d'extraire ses points forts et ses faiblesses. L'usage de patron peut intervenir pour mettre cette information en évidence.

1.1.2 Autres considérations

Différentes contraintes sont utilisées pour optimiser l'ordonnancement. Demiroz et al. [26] optimisent le temps d'exécution et le nombre de processeurs. D'autres auteurs [76], [77], [78], [79] maximisent à la fois le temps d'exécution et l'énergie utilisée par la modulation de la tension. Switalski et Seredynski [80] utilisent la variation de tension avec la variation de fréquences dans le but d'obtenir des ordonnancements qui consomment le moins. D'autres spécialistes [81], [82], [83] tiennent compte des pannes qui peuvent survenir sur le réseau multiprocesseur. En pratique, le coût de communication varie selon l'architecture, la fréquence d'opération et la possibilité d'extension du lien. La densité d'un GAD est définie par rapport au nombre d'arcs versus le nombre de nœuds. Le ratio communication versus exécution CCR d'un GAD est obtenu par la moyenne des poids de chaque arc sur la moyenne des poids des temps d'exécution. Ce ratio donne la dominance pour savoir si le GAD est plus

en mode communication ou en mode exécution. Nous avons vu les principaux groupes, les heuristiques et les aléatoires.

Comme mentionné plus haut, dans la littérature, il existe presque autant d'algorithmes que d'applications. L'analyse des algorithmes est faite avec des métriques variées et donne des résultats différents selon les hypothèses. Pour ces raisons, la documentation est devenue importante dans le processus de design. Capilla et al. [84] montrent qu'il y a un manque d'outils pour l'enregistrement et l'exploitation des décisions de conception architecturales, menant à des coûts de maintenance élevés et des pertes de connaissance à propos des décisions prises qui auraient pu être bénéfiques pour leur réutilisation. Ils ont proposé un système de support de décision qui traite de ces questions en définissant la connaissance architecturale pertinente avec les contraintes d'attributs obligatoires et facultatifs. En outre, les modèles sont utilisés comme des ensembles de décisions de conception. Ces considérations peuvent servir à réutiliser et partager les connaissances architecturales comme expliqué par Lago et Avgeriou [85] qui traitent des méthodes et des outils qui peuvent extraire, découvrir et partager les connaissances. Keutzer et Mattson [86] affirment que l'aide d'un langage de modèle de conception pour les logiciels d'ingénierie conduit à un logiciel parallèle de haute qualité et la conception de logiciels robuste. Dans cette perspective, les techniques d'ordonnancement peuvent être vues à la fois comme une stratégie d'algorithme et un patron. L'ordonnancement peut alors être considéré comme un élément obligatoire. Nous devons être capables de choisir un algorithme d'ordonnancement et de conserver les traces de ce choix avec toutes les informations spécifiques qui nous a guidés vers la décision.

Guidés par les considérations précédentes, nous proposons un environnement qui utilise des langages de modèle et des règles d'association pour modéliser et documenter les algorithmes d'ordonnancement. La prochaine section couvre les différentes façons de représenter des connaissances. Notre but est de modéliser et d'accroître les connaissances des algorithmes d'ordonnancement.

1.2 La problématique de la représentation des connaissances

Cette section couvre les différentes recherches pour la représentation du savoir à l'aide de patrons. Alexander [87] présente le concept de modèle appliqué à un problème d'architecture urbaine. Il explique comment un problème récurrent dans la construction et l'environnement dans une ville peut être un motif, par exemple relatif à la représentation de la construction d'une porte. Il montre comment une collection de patrons peut créer un ensemble de modèles. Ainsi, un groupe de patrons de portes représenterait différentes façons de construire une porte. De plus, un ensemble de patrons peut être lié à un autre patron. Un langage de patrons formalise ces opérations. En d'autres termes, il s'agit d'une méthode structurée pour exprimer des connaissances sur une ou plusieurs solutions éprouvées à un problème récurrent.

En 1987, ces concepts ont été transposés à la conception de logiciels. Beck et Cunningham [88] ont appliqué un patron à la conception d'une interface utilisateur. Depuis lors, les patrons ont été utilisés dans différents domaines. Roberts et al. [89] utilisent des patrons pour capturer l'architecture logicielle. Gordon [90] utilise la connaissance des patrons pour fournir à un professionnel une idée claire et concise d'un ensemble de logiciels d'architecture. En outre, un patron fournit aux utilisateurs une expérience de cas d'affaires distinctifs et des attentes de travail existant sur l'architecture logicielle. Zdun [91] propose une méthode pour trouver et classer systématiquement les patrons de différentes sources. Il a construit son approche de façon à ce qu'elle soit compatible avec différentes grammaires et les modèles linguistiques avec l'analyse de l'espace de conception. Celle-ci repose sur la manipulation du format des patrons distincts et leur transformation en une seule grammaire et langage. En outre, le procédé a introduit la notion de patron, des séquences de patrons et un diagramme de séquence avec les patrons. Harrison et al. [92] ont proposé d'utiliser des patrons pour capturer les décisions architecturales : les modèles architecturaux permettent un enregistrement de l'information et une meilleure compréhension de la justification et des conséquences des décisions prises par le développeur du logiciel.

Gordon [90] utilise les patrons pour documenter structurellement un ensemble d'architectures logicielles de façon claire et précise tout en fournissant plus de connaissances pour les professionnels de l'informatique. Les patrons peuvent donner une solution à une

problématique récurrente tout en donnant l'expectative des résultats obtenus en se basant sur les réalisations antérieures. Le concepteur possédant les connaissances des patrons pourrait choisir un patron en particulier pour ses besoins. Zdun [91] propose une méthode pour trouver et classer systématiquement les patrons de différentes sources pour les objets distribués par intergiciel. Sa méthode consiste à chercher un patron qui résout un problème particulier et à utiliser une grammaire de langage de patrons avec une analyse de l'espace de design. La grammaire peut être utilisée pour manipuler des patrons et permet de traiter des séquences de patrons. Elle permet de définir et de représenter les options, les exigences ou les variations des patrons pour un problème donné. Par exemple, la séquence <Patron A, Patron B, Patron C> peut représenter trois patrons alternatifs, trois solutions différentes, pour le même problème. Cette opération pourrait être effectuée par une grammaire flexible qui extrairait des connaissances à partir des patrons. L'analyse de l'espace de conception saisit la logique du design. La conception peut être structurée avec une représentation explicite de la raison pour laquelle on a choisi une solution et toutes les options qui pourraient être utilisées.

Zhu et al. [93] introduisent le concept de modèles d'exploration de données pour améliorer et soutenir l'évaluation des architectures logicielles, le but étant d'utiliser un modèle de qualité pour améliorer la qualité et de prévoir les attentes pour une solution avec des paramètres différents à l'aide de la conception orientée modèle. Selon certains auteurs [91], [93], nous devons déterminer par une méthode qualitative la performance d'un patron pour pouvoir déterminer sa valeur pour rapport à d'autres patrons ; nous proposons donc la notion de la qualité d'un patron. Santos et Koskimies [94] ajoutent une couche supplémentaire de modules réutilisables comme patron. Cela a facilité le développement des interfaces réutilisables et a permis de créer une abstraction plus élevée des interfaces tout en créant des applications de référence. Borchers [95] propose une approche pour capturer les connaissances des développeurs et des experts du domaine de l'application dans le développement de logiciels, l'interaction humain-ordinateur et le domaine d'application. Roberts et autres [89] montrent comment les modèles de conception peuvent être utilisés dans des éléments logiciels réutilisables. Keutzer et Mattson [86], de leur côté, proposent le langage *Our Pattern Language* (OPL) et cinq catégories de modèles de conception :

structure, calcul, stratégie algorithmique, mise en œuvre et modèles d'exécution parallèles. Ils ont également présenté un modèle pour le parallélisme de tâches dans les catégories de la stratégie de l'algorithme, le modèle parallèle embarrassant. Enfin, Borchers [95] montre que la réalisation d'une exposition interactive de musique peut être représentée sous forme de patrons.

Les études antérieures ont défini de multiples langages-patrons et les ont utilisés pour augmenter et faciliter la connaissance dans divers domaines. Certains auteurs ont également présenté des grammaires pour manipuler des modèles et effectuer de l'extraction de données afin d'obtenir des informations précieuses sur des groupes de motifs en vue de résoudre des problèmes particuliers. Toutefois, les grammaires, l'extraction de données et les groupements diffèrent entre les chercheurs et il n'existe pas de pratique constante.

De plus, dans un contexte général, la documentation est devenue un enjeu important dans le processus de design. Comme indiqué ci-dessus, les patrons peuvent être utilisés dans différents domaines. Santos et Koskimies [94] indiquent qu'il existe un manque de méthodes et d'outils pour enregistrer et exploiter les décisions de conception architecturale. Capilla et al. [84] proposent un système d'aide à la décision de conception par rapport au fait d'avoir le bon algorithme avec une architecture ciblée. Comme l'indiquent Keutzer et Mattson [86], les techniques d'ordonnancement peuvent être une stratégie et sous forme de patron stratégique. Capilla et al. [84] définiraient alors les connaissances pertinentes architecturales avec leurs contraintes d'attributs obligatoires et facultatifs. L'ordonnancement peut être considéré comme un élément obligatoire; en effet, la raison d'être est de savoir pourquoi nous utilisons cet algorithme et les informations reliées à cette décision. En outre, ces auteurs [84] utilisent des motifs comme un ensemble de décisions de conception. Ces considérations peuvent être utilisées pour réutiliser et partager les connaissances architecturales comme l'ont expliqué Lago et Avgeriou [85], qui traitent des méthodes et des outils qui peuvent extraire, découvrir et partager les connaissances.

1.3 Sommaire et contribution

La recherche sur l'ordonnancement des GADs a principalement eu comme objectif de minimiser le temps maximal d'exécution. De plus, chaque algorithme est pensé en fonction d'une architecture, d'une application ou une caractéristique donnée. Dans ce contexte, on dispose d'une bibliothèque d'algorithmes qui couvrent diverses architectures et technologies.

Plusieurs chercheurs proposent des algorithmes génériques qui s'appliquent à un grand nombre d'applications, mais qui posent la question de leur efficacité dans tous les cas. Récemment, les concepteurs d'algorithmes ont noté que le fait de segmenter le GAD en morceaux réduisait l'impact des métriques sur des paramètres tels le temps d'exécution estimé d'une tâche sur un processeur. Par ailleurs, la façon de comparer les résultats de différents algorithmes mène à différentes conclusions selon les critères de comparaison. Certains auteurs ont élargi l'ordonnancement à plusieurs variables en ajoutant le nombre de processeurs en plus du temps maximal d'exécution. Dans la littérature, nous ne trouvons pas de méthode qui détermine l'algorithme à utiliser pour ordonnancer un design particulier. La plupart des approches reportées font des comparaisons de performances par rapport à d'autres, mais elles ne donnent pas pour autant le type de graphe qui est le mieux adapté pour un design arbitraire.

De nombreux algorithmes d'ordonnancement ont été présentés. Les concepteurs d'algorithmes comparent généralement leurs algorithmes à ceux des autres par le biais de mesures statistiques et de quelques exemples de conceptions [13], [19], [24], [17]. Mais les algorithmes d'un groupe ont leurs propres objectifs en matière d'optimisation et ne sont souvent comparés qu'aux autres membres du groupe. Un algorithme d'ordonnancement de type liste sera ainsi comparé aux algorithmes d'ordonnancement de type liste, un algorithme génétique sera comparé aux autres algorithmes génétiques, etc. Cependant, le fait que les algorithmes et autres architectures parmi lesquels il faut choisir sont nombreux, que chaque algorithme dispose de ses propres paramètres et qu'il peut être divisé en sous-algorithmes (définis en tant que variations dotées de diverses propriétés et puissances), rend complexe le choix de celui qui est le plus adapté à une application. Différents travaux suggèrent de prévoir des boîtes à outils d'ordonnancement [96] qui se focalisent sur un nombre limité

d'algorithmes, sachant que le seul moyen de sélectionner le plus approprié pour une application donnée est de les essayer tous et de comparer les résultats. Les nombreux algorithmes devant être évalués et une méthodologie de sélection plus efficace étant requis, cette approche nécessite des calculs intensifs.

Dans ce travail, nous proposons une méthode générale pour l'ordonnancement de tâches qui est basée sur l'exploitation de connaissances historiques et qui tient compte des métriques d'un design et vise l'optimisation d'un ensemble de paramètres. L'architecture sous-jacente tient compte du type de système, des algorithmes disponibles et du type d'application. La méthode est basée sur la décomposition, l'analyse et la résolution de problème dans la partie gauche de la méthode Vee et de la technique de base d'analyse par compromis [12], [97]. Elle se résume ainsi : dans un premier temps, le concepteur fournit à l'outil une représentation du projet; celui-ci générera alors un ordonnancement qui répond aux critères énoncés. La nouveauté de la méthode repose sur un sélecteur d'algorithme qui détermine un patron du graphe et va choisir l'algorithme le plus approprié tout en utilisant des patrons de conception, tirés de la fouille de données et des méthodes de gestion de projet [12], [97]. La notion de patrons de conception sera utilisée avec un langage de patrons [95]. La plateforme proposée repose sur la création d'un référentiel de travail unique et des connaissances validées provenant de différentes sources. Les sources peuvent venir d'articles, de simulations ou d'analyses. L'environnement utilise le concept de modèle pour représenter un algorithme d'ordonnancement en fournissant des informations sur ses forces et ses faiblesses dans un dossier de définition.

L'utilisation de la fouille de données aidera à obtenir une vue d'ensemble des algorithmes sans avoir à effectuer d'analyse statistique sur un grand nombre de cas. Elle permet entre autres d'utiliser les connaissances brutes et d'extraire de nouvelles connaissances. De plus, une nouvelle méthode sur la cartographie des graphes sera mise en œuvre; cette nouveauté permet de représenter de grands graphes dans un espace réduit, donnant ainsi une vue d'ensemble du design au concepteur. En plus des nouvelles connaissances, de nouvelles approches de documentation vont être présentées.

Nous proposons donc de réunir les règles d'association, les langages de patrons et les grammaires pour modéliser et faciliter l'exploitation de l'algorithme d'ordonnancement avec une méthodologie cohérente. Nous proposons un nouveau modèle de données pour les algorithmes de planification, une méthode pour créer des modèles de planification personnalisée qui introduit également la notion de séquences de motifs pour les algorithmes d'ordonnancement. En outre, nous proposons quelques concepts sur la représentation des connaissances de l'ordonnancement qui ne sont pas liés directement aux motifs.

En résumé, nous nous concentrons sur un nouveau moteur de gestion des connaissances qui peut répondre à plusieurs problématiques d'ordonnancement, aidant ainsi les concepteurs à réaliser leurs applications et les experts à élaborer des stratégies sur des algorithmes d'ordonnancement.

Le chapitre suivant donne un aperçu général de l'environnement et de son opération.

CHAPITRE II

VERS UNE AUTOMATISATION DE L'ORDONNANCEMENT DES TACHES

Ce chapitre reprend une partie d'article publié dans les actes de *International Conference on Microelectronics* (ICM'2010) [98]. Il décrit une nouvelle méthodologie de représentation des connaissances des techniques d'ordonnancement et explore l'utilisation et la possibilité d'inclure cette méthodologie dans le cadre de développement de systèmes. Il présente une nouvelle perspective de représentation des données des algorithmes d'ordonnancement. Y est abordée l'importance d'une nouvelle vision des données sur les algorithmes d'ordonnancement, en particulier la proposition d'exploiter les données brutes d'un GAD d'application comme complément d'information à l'analyse statistique. De nouvelles techniques d'acquisition et d'utilisation des données brutes sont présentées et celles-ci facilitent l'usage des outils automatisés tout en permettant à un non-expert de faire de l'exploration de design. En particulier, l'usage de l'analyse formelle de concepts ouvre la porte à de nouvelles possibilités d'organisation des connaissances et peut être employé par l'usager qui n'est pas spécialiste et par l'expert des algorithmes de planification des tâches. Les propositions telles que l'analyse formelle de concepts sont considérées à l'intérieur d'une nouvelle plateforme. Celle-ci permet, entre autres, de trouver un algorithme efficace pour le designer ou l'expert qui veut créer et optimiser ses algorithmes d'ordonnancement.

Dans un premier temps, la section qui traite de l'expert montre comment modéliser un ensemble d'algorithmes sous forme de diagramme de Hasse. Dans un deuxième temps, la section qui traite de l'usager qui veut choisir un algorithme élabore la nouvelle méthode d'organisation des connaissances. Un système de deux filtres est proposé en utilisant la méthode Vee [12] pour aider le concepteur à choisir l'algorithme, même s'il n'est pas spécialiste du domaine d'ordonnancement. La nouvelle plateforme de connaissance permet donc d'acquérir et de présenter les algorithmes efficacement.

2.1 Méthodologie proposée

Notre propos est de trouver l'algorithme d'ordonnancement le plus opportun pour le concepteur. L'idéal serait d'avoir une méthode d'évaluation des performances efficace, neutre et rentable pour l'ensemble des différentes applications.

2.1.1 Environnement

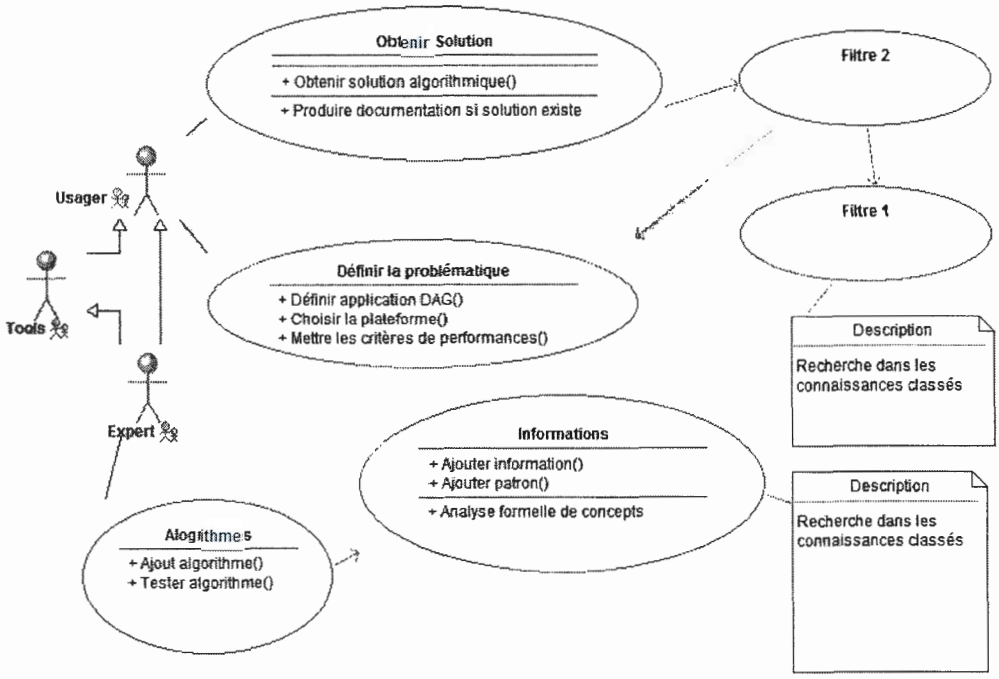
La méthodologie proposée prend en considération aussi bien les contraintes du concepteur que celles imposées par la technologie, l'impact métrique et l'ordonnancement des tâches. Elle fonctionne comme suit : après avoir créé une représentation graphique de sa conception, le concepteur définit les entrées et sorties, l'architecture de mise en œuvre ainsi que les contraintes requises par le système telles que la marge de défaillance, les limites des processeurs et leurs vitesses (l'association de multiples algorithmes à plusieurs GAD est également possible). Ces informations permettent de trouver l'algorithme d'ordonnancement approprié au GAD donné, ainsi que le montrent les explications suivantes.

Les algorithmes d'ordonnancement disponibles sont créés par des experts qui savent quelles plateformes utiliser en fonction de leurs particularités (par exemple la fréquence de fonctionnement, la tension d'alimentation, etc.). Les algorithmes sont habituellement développés pour optimiser des types d'architecture et d'autres paramètres spécifiques; les interfaces frontales « Expert » et « Utilisateur » permettent de gérer et d'exploiter ces informations. L'utilisateur est une personne qui souhaite mettre en œuvre une application et utilise pour ce faire un environnement de développement intégré (IDE). L'expert est, quant à lui, responsable de l'optimisation et de l'ordonnancement des applications. Il ou elle utilise l'interface frontale « Expert » afin de modifier et mettre à jour la base de données des algorithmes relatifs aux IDE dont les utilisateurs disposent. Les deux paragraphes suivants présentent les interfaces frontales « Utilisateur » et « Expert ».

2.1.2 Interface frontale « Expert »

Une interface « Expert » permet d'acquérir et d'organiser les informations relatives aux algorithmes et à la technologie. Elle sert à tester et à valider les algorithmes avant leur inclusion dans une base de données structurée disponible pour l'interface frontale « Utilisateur ». La figure 2.1 montre une version simplifiée de l'interface.

Figure 2.1 : Version simplifiée de l'interface frontale « Expert »



Ajouter des informations à cette interface peut présenter multiples facettes. Nous nous concentrons essentiellement, dans le présent chapitre, sur l'acquisition d'algorithmes d'ordonnancement. La figure 2.1 détaille l'organigramme de l'interface frontale « Expert » correspondante. L'expert utilise cette interface pour entrer des informations concernant les algorithmes d'ordonnancement. Ces informations brutes sont structurées en patrons de logiciels pouvant être appliqués pour résoudre certaines catégories de problèmes et sont décrites en langage de schéma détaillant leurs structures et contenus. Nous avons défini un

langage de schéma pour les algorithmes d'ordonnancement, en commençant par la sémantique proposée en [95] (où le mécanisme est actionné manuellement). Nous adaptons par conséquent la structure et contenu d'un modèle de conception à l'acquisition des algorithmes d'ordonnancement. Les informations brutes émanant des experts peuvent comprendre le nom, le rang, l'ensemble de problèmes applicables, la puissance et des exemples de déploiement. Le rang définit habituellement la validité de l'information (par exemple un seuil de confiance comme il sera vu plus loin), mais il peut également servir de critère de couplage. Ainsi, si plusieurs algorithmes s'avèrent compatibles avec une application, un rang plus élevé indique un algorithme doté d'un degré de compatibilité plus important avec le problème.

Afin d'automatiser le processus, les caractéristiques appropriées de chaque modèle décrit dans le langage de schéma sont mémorisées dans une base de données. Par exemple, si A a été testé afin d'être utilisé dans une plateforme de 2 à 16 processeurs et de 2 à 100 tâches, cette information servira éventuellement d'indicateur de puissance. Le tableau 2.1 figurant ci-dessous illustre ce processus.

Tableau 2.1

Représentation tabulaire d'un ensemble d'algorithmes

	A	B	C	D	E	F
A1	X		X		X	X
A2		X		X	X	
A3	X			X		
A4	X		X		X	

(A,B(CCR), C(Système hétérogène),D(Système homogène),E(Nombre de tâches), F(2 processeurs))

Les rangées A_1 à A_4 correspondent à quatre algorithmes d'ordonnancement potentiels, chacun d'entre eux pourvu de propriétés différentes indiquées dans le contenu des colonnes A à F. Les colonnes A et B représentent ici respectivement CCR < 1 et CCR entre 1 et 5; C indique

un système hétérogène, D un système homogène, E un système hétérogène; E montre une capacité de 2 à 100 tâches et F désigne deux processeurs. La création d'un tableau identique pour chaque plateforme cible permettra de synthétiser les informations relatives aux algorithmes d'ordonnancement disponibles. Son but est d'utiliser ces informations et d'aider l'utilisateur à sélectionner l'algorithme (à partir d'un ensemble d'algorithmes) et la plateforme adéquats sans qu'il connaisse les spécificités des algorithmes d'ordonnancement ou de la plateforme ciblée.

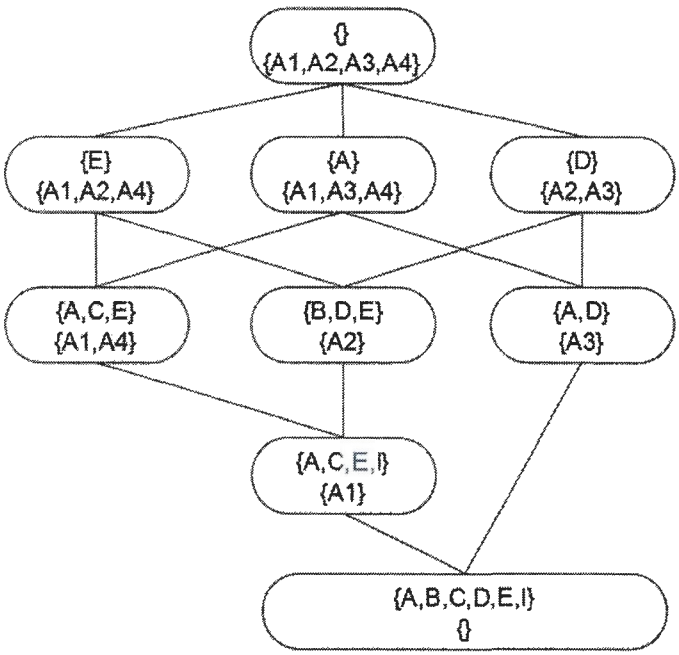
Les algorithmes se rapportent généralement à des architectures spécifiques et peuvent disposer de sous-algorithmes pour certaines applications particulières (nous considérons les sous-algorithmes comme des algorithmes différents afin de simplifier les choses). L'utilisateur peut donc avoir à choisir parmi des milliers de possibilités algorithmiques dotées de nombreux paramètres technologiques et liés à l'application. Cette opération requiert le regroupement et l'ordonnancement des informations.

L'analyse formelle de concepts (FCA) permet de structurer la variété algorithmique disponible en groupes conceptuels. Par exemple, un groupe d'algorithmes et un groupe de concepts peuvent amener à définir deux groupes conceptuels, l'un pour les systèmes homogènes et l'autre pour les systèmes hétérogènes. Nous créons alors un treillis de concepts avec l'algorithme Bordat [99] qui tisse les liens entre des éléments issus de relations binaires. Un résumé de FCA suit : concrètement, un concept C est un triple composant qui comprend un objet F, un attribut P et une relation R établie entre eux. Par conséquent, C = (F, P, R). Une paire (F, P) forme un concept si $T_1(F) = P$ et $T_2(P) = F$, où T1(F) est le sous-ensemble de P dont les éléments sont en relation avec les éléments de F et $T_2(P)$ le sous-ensemble de F dont les éléments sont en relation avec ceux de P. Un concept représenté par une paire (F, P) est un nœud; F est appelé « extension » et P « intention » du concept; ils sont duaux. Un ensemble de nœuds est un treillis de concepts L.

Ainsi, dans le Tableau 2.1, L compte 9 concepts parmi lesquels se trouve celui reliant les attributs (A, C, E) aux algorithmes (A_1, A_4), comme le montrent les entrées surlignées ou, plus clairement encore, le graphe conceptuel (Diagramme de Hasse) de la figure 2.2. Un

treillis de concepts est créé pour tous les algorithmes d'ordonnancement et les graphiques conceptuels correspondants produits permettent de structurer les concepts qui généralisent et spécifient les relations entre les algorithmes. Chaque nœud de la figure 2.2 correspond donc à un concept doté d'une liste d'algorithmes et d'une liste d'attributs.

Figure 2.2 : Diagramme de Hasse d'un ensemble d'algorithmes d'ordonnancement

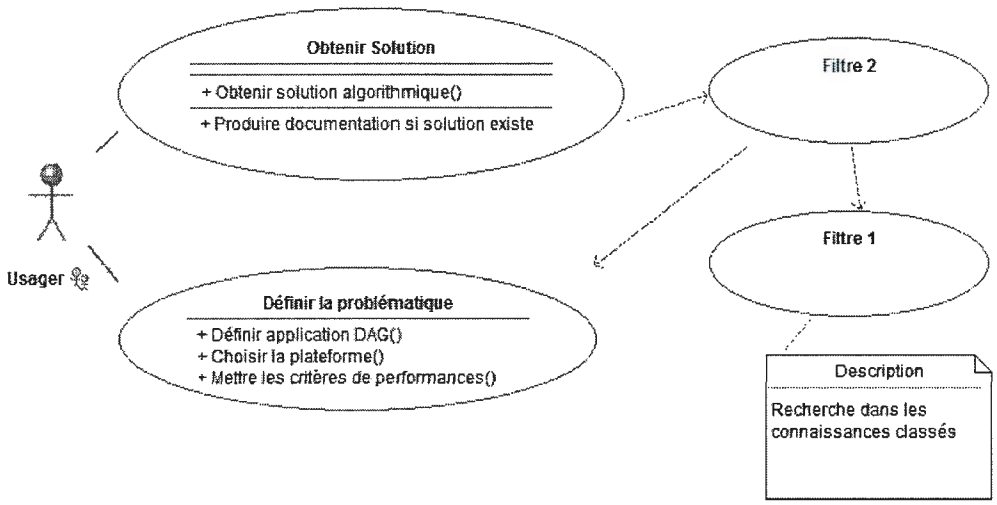


Ces informations sont classées puis enregistrées et l'interface frontale « Utilisateur » les exploite afin d'apporter les solutions d'ordonnancement. La section suivante donne les détails.

2.1.3 Interface frontale « Utilisateur »

L'interface « Utilisateur » permet à l'utilisateur non spécialiste en algorithmes d'ordonnancement de profiter d'informations déjà emmagasinées relatives à l'ordonnancement.

Figure 2.3 : Version simplifiée de l'interface frontale « Utilisateur »



La figure 2.3 détaille l'organigramme de l'interface frontale « Utilisateur ». L'utilisateur (ou outil de conception) entre une architecture et un GAD à employer dans le cadre de la conception, mais également une liste de paramètres représentant les exigences de l'application telles que la marge de défaillance, les limites des processeurs et leurs vitesses. Un algorithme répondant aux besoins énoncés lui est alors suggéré grâce à l'exploitation des contenus de la base de données dans laquelle se trouvent les informations classées. Pour ce faire, une première étape consiste à extraire les caractéristiques du GAD et à identifier les algorithmes d'ordonnancement compatibles dans la base de données. Deux étapes de filtrage basées sur des analyses de décomposition, des processus de résolution [12] et une exploration conceptuelle [97] permettent ensuite de sélectionner l'algorithme définitif.

Filtre 1 transforme les caractéristiques du GAD, l'architecture et les contraintes en liste d'attributs. Un diagramme de Hasse est alors généré pour extraire un groupe conceptuel GA' qui tire ses attributs de l'ensemble GA des algorithmes disponibles. Donc, si le graphique compte 25 tâches, que CCR < 1 et qu'une plateforme hétérogène est utilisée, le filtre commence par trouver chaque algorithme susceptible de traiter 25 tâches et pouvant être utilisé sur une plateforme hétérogène. Le concept correspondant dans le Tableau 2.1 à la page 30 et la Figure 2.2 est (A, C, E); (A_1, A_4) et GA' = {A_1, A_4}. Cette phase de filtrage comprend également un antipatron qui permet d'exclure les algorithmes non applicables au problème de conception donné, ceci grâce aux informations relatives à la faiblesse des algorithmes et à la vérification des antipatrons associés. À la suite de cet élagage, GA' deviendra GA".

Filtre 2 fourni des mesures telles que le temps d'exécution avec certaines métriques de mesures et de modèle-pour chaque algorithme candidat dans GA" (A_1 et A_4 dans l'exemple précédent). Différentes estimations sont effectuées afin d'établir les valeurs médianes ainsi que les pires en matière de coût de calculs. L'utilisateur peut se servir de ces mesures pour sélectionner l'algorithme le plus adapté à sa conception. L'algorithme sélectionné offre à l'utilisateur ou à l'outil de conception la meilleure solution qui soit pour mettre en œuvre la conception.

L'algorithme utilisé est inspiré du modèle en V pour le développement de systèmes [12]. Les modèles en V tels Vee, Vee+, Vee++ sont des modèles de développements logiciels séparés en une séquence de phase de développement. Le modèle en V comporte une partie droite et une partie gauche. La partie gauche sert principalement à la spécification et les décompositions de diverses composantes avec le maximum de détails. La partie gauche illustre la partie intégration, l'assemblage, des composants et des vérifications. Les modèles peuvent être évolutionnaires en procurant une amélioration de la solution par le livrable de plusieurs versions. Ces modèles peuvent aussi servir dans un contexte d'un ou plusieurs livrables.

Vee est le premier modèle de base développé en 1987 par la NASA. En 1990, Vee+ ajoute la notion des actionnaires et utilisateurs qui sont impliqués dans le processus de

développement avec les notions d'opportunités, de gestion de risques et de résolution de problèmes de vérification.

En 1991, Vee++, ajoute à Vee+ des processus pour avoir une représentation compréhensible des processus d'intégration qui font partie typiquement du développement système. Cet ajout prend la forme de trois dimensions. L'axe vertical est le temps présent, l'axe horizontal est le temps de réalisation ainsi que la maturité du développement. Le dernier axe comporte l'intersection entre plusieurs processus de développement.

Par conséquent, un ingénieur système utilise le modèle Vee pour expliquer les différentes approches prises lors de la réalisation de projets tout en mentionnant les conséquences de celles aux non-initiés. La représentation, la conservation des informations concernant le développement sont importantes et est prise en compte dans le modèle Vee. Dans ce sens, l'interface usager doit être capable de mettre en évidence les choix algorithmes pris.

2.2 Complexité de la mise en œuvre de la méthodologie

La mise en œuvre de la méthodologie est essentiellement attribuable aux interfaces frontales « Expert » et « Utilisateur » et au fait que l'interface frontale « Expert » gère l'acquisition d'informations. Nous utilisons l'algorithme Bordat pour créer un treillis de concepts [99] [100] qui a une complexité $O(n^2m^2 2^k)$, où n correspond au nombre d'algorithmes, m au nombre d'attributs et k au nombre maximum d'attributs par objet.

L'interface frontale « Utilisateur » se sert des filtres 1 et 2. La complexité de Filtre 1 réside dans le nombre de tâches ou d'attributs utilisés par le concepteur; Filtre 2, quant à lui, n'utilise que les algorithmes candidats dans le groupe GA", au lieu de tester la totalité des algorithmes. Le degré de complexité en découlant dépend des candidats sélectionnés.

2.3 Sommaire

L'ordonnancement et les problèmes d'attribution sont bien connus. Leur complexité s'avère malheureusement exponentielle. Les algorithmes disponibles utilisent des estimations de

valeurs telles que la valeur médiane, le pire et le meilleur des cas pour évaluer les coûts de calculs et de communication. Ces valeurs estimées sont imparfaites, voire approximatives, et affecteront de ce fait les résultats algorithmiques obtenus. Les architectures ainsi que le type d'application sont également susceptibles d'exercer une influence. Nous avons par conséquent besoin d'un algorithme généraliste avec des performances constantes, peu importe le modèle utilisé pour représenter l'architecture. Nous avons décrit dans ce chapitre le recours à une analyse formelle de concepts en tant qu'outil d'analyse de haut niveau pour réduire la complexité du problème d'ordonnancement et l'impact des valeurs estimées.

Notre approche ne repose pas sur l'utilisation de statistiques, mais sur des analyses d'informations provenant de données brutes liées aux applications individuelles. Nous avons adopté un environnement flexible qui exploite les informations technologiques et architecturales classées pour les applications d'ordonnancement sur multiprocesseurs SoCs, privilégiant une approche de conception descendante. Les chapitres suivants fournissent des exemples d'utilisation et décrivent le langage de patrons qui permet l'apprentissage du savoir sur les algorithmes d'ordonnancement avec les règles d'association pour exploitation subséquente. Mais avant, le prochain chapitre s'attaque au problème de l'extraction des connaissances à partir des informations disponibles en vue de construire les règles d'association.

CHAPITRE III

APPRENTISSAGE PAR RÈGLES D'ASSOCIATION POUR L'EXPLOITATION DES CONNAISSANCES SUR LES PERFORMANCES D'UN L'ALGORITHME D'ORDONNANCEMENT

Pour donner suite au chapitre précédent, une nouvelle approche de forage de données sur les algorithmes d'ordonnancement est proposée ici. Une partie d'article publié dans les actes de *International New Circuits and Systems Conference* (NEWCAS'2011) [101] est reprise. Une nouvelle approche basée sur l'utilisation des techniques d'apprentissage de règles d'associations y est suggérée. En combinaison avec le forage de données, elle offre de nouvelles possibilités pour trouver des informations pertinentes sur les performances algorithmiques pour l'expert en ordonnancement. Nous montrons l'importance de trouver les forces et les lacunes d'un algorithme d'ordonnancement. Une méthodologie d'extraction de données est proposée avec une méthode de comparaison entre deux algorithmes. Dans un premier temps, une introduction au forage de données est présentée. Ensuite, une proposition de représentation binaire des informations est brossée. Cette représentation permet l'usage de forage e: l'utilisation de règles associatives afin d'extraire les connaissances disponibles sur les capacités d'un algorithme d'ordonnancement. La méthodologie présentée offre de nouvelles perspectives sur le repérage des forces et lacunes d'une technique d'ordonnancement et facilite l'exploitation des données brutes. L'utilisation de deux algorithmes différents permet d'illustrer l'extraction de nouvelles connaissances. De plus, la proposition d'utiliser la différence de règles d'association permet de comparer deux algorithmes selon la signature d'un graphe.

Ce chapitre examine également la façon d'extraire efficacement des informations quant à la performance d'un algorithme d'ordonnancement dans le cadre d'un ensemble d'applications par l'apprentissage des règles d'association entre les attributs des applications et les mesures de performances des applications. La nouvelle méthodologie présentée permettra à la fois d'accroître les connaissances du concepteur à propos d'un algorithme d'ordonnancement

particulier, mais aussi de comparer les algorithmes. Enfin, la nouvelle méthode d'extraction et de comparaison des connaissances concernant les algorithmes d'ordonnancement conduit à une nouvelle perspective sur la sélection de ces algorithmes, en donnant un aperçu sur leurs capacités et en permettant des comparaisons entre eux pour des attributs d'application donnés. Elle s'appuie sur la recherche de règles d'association [102] pour des mesures de performances données.

Le chapitre commence avec la méthodologie d'extraction des connaissances sur un algorithme, puis expose les résultats expérimentaux pour deux techniques d'ordonnancement que nous avons testées.

3.1 Méthode proposée

Au dernier chapitre, nous avons proposé un environnement automatisé qui assiste le concepteur pour sélectionner l'algorithme d'ordonnancement à partir d'un ensemble d'alternatives. Cet environnement s'appuie sur la création d'un référentiel de connaissances uniformément formatées et validées à partir de différentes sources. Les sources doivent provenir de documents, de simulations ou bien d'analyses. Le patron logiciel peut être utilisé pour représenter les forces et faiblesses d'un algorithme. Ici, nous suggérons d'utiliser l'apprentissage de règles d'association afin d'extraire ces informations, donnant ainsi des indications précieuses sur les techniques d'ordonnancement au concepteur. Ces connaissances sont destinées aux utilisateurs expérimentés qui créent et possèdent des connaissances sur l'algorithme d'ordonnancement et l'architecture.

3.1.1 Exploration des règles d'association

Nous définissons dans un premier temps les concepts utilisés dans ce travail. Soit un ensemble d'attributs binaires $A = \{a_i\}_{i=1,2...,N}$ désignant un modèle; une règle d'association est une implication $x \Rightarrow y$, où x et y sont des sous-ensembles disjoints de A. En d'autres termes, la présence de certains attributs dans le modèle est concomitante avec celle d'autres attributs, dans notre cas pour la recherche d'indicateurs des points forts et des faiblesses.

Le niveau de confiance en une règle (*Confidence*) est la mesure relative du nombre de fois où le résultat de la règle est vrai lorsque l'antécédent est vrai :

$$Confidence(x \Rightarrow y) = \frac{Support(x \cup y)}{Support(x)}$$

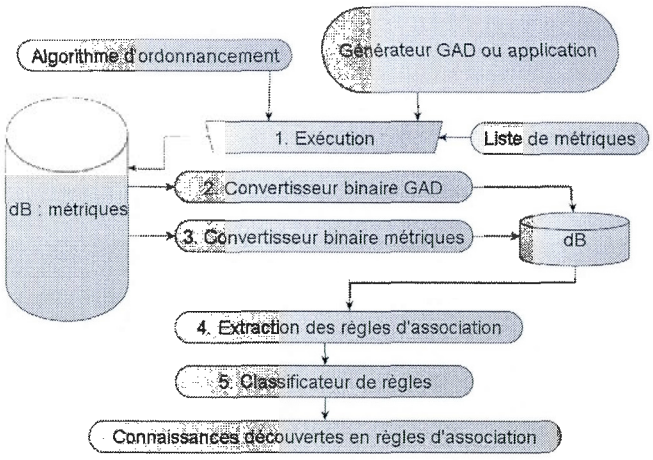
Pour un ensemble de modèles P = $\{p_1, p_2, \ldots, p_M\}$, chacun caractérisé par un ensemble d'attributs spécifiques A_i; Support(x) est la fréquence relative de x dans P. Ainsi, la confiance ($x \Rightarrow y$) peut être considérée comme l'estimation de la probabilité conditionnelle de y étant donné x. Il existe des règles d'association fréquentes et rares [103] en fonction d'un seuil de support minimal.

Le lift d'une règle d'association compare sa fréquence relative observée par rapport à celle attendue en supposant des termes indépendants, antécédents et suivants. Il est défini par :

$$Lift(x \Rightarrow y) = \frac{Confidence(x \Rightarrow y)}{Support(y)}$$

Nous proposons d'utiliser l'apprentissage de règles d'association pour extraire les points forts et les faiblesses d'un algorithme. La figure 3.1 résume la méthodologie.

Figure 3.1 : Découverte de connaissances à l'aide d'extraction de règles d'association



Il existe cinq étapes pour découvrir les règles d'association pour un algorithme d'ordonnancement :

1. Créer une base de données des applications, composée des GAD produits de façon pseudo-aléatoire [13] avec des définitions d'attributs, comme décrit au chapitre précédent, et des mesures appropriées calculées après avoir appliqué l'algorithme d'ordonnancement pour chaque GAD. Les informations obtenues (attributs du GAD plus les mesures des performances) pour chaque application sont stockées comme éléments d'information.

2. Pour chaque attribut de GAD, créer une séquence binaire où chaque position de bit représente la présence ou absence de valeurs données ou de plages de valeurs d'un attribut, de manière à générer une représentation binaire de la signature du GAD. Pour cela, les attributs du GAD sont considérés comme pris à partir d'un ensemble d'ensembles Ri, chacun représentant les valeurs possibles – singletons ou domaines – pour un attribut. Par exemple, si la signature du GAD est E = (N, C, α, β, D), où N est le nombre de nœuds dans le GAD, CCR le rapport communication-calcul, α le taux de parallélisme, β la portée du calcul pour chaque nœud et D la densité du GAD, alors avec la référence R = (RN, RC, Rα, Rβ, RD) qui fournit le domaine de chacun des attributs du GAD, une signature de de E peut être créée et signaler la présence et l'absence de sous-domaines relatifs de Ri (voir exemple plus bas).

3. Répéter l'étape 2 avec les mesures calculées à l'étape 1 afin d'avoir les métriques et les signatures de chaque GAD. Si une métrique est positive, nous avons une forme de résultats binaires positifs (patron) ou si la métrique est négative, nous parlons d'anti-patrons.

4. Obtenir les règles d'association des séquences obtenues lors des étapes 2 et 3. Pour cela, nous employons l'outil Rapid Miner [11], qui utilise l'algorithme fp-growth[1] [104] pour trouver les ensembles d'éléments appropriés à partir des données et, ensuite, crée les règles d'association (nous utilisons 75 % de confiance pour les éléments de bloc pour la réalisation de l'exploration des données).

5. Après l'exploration des données, un classificateur trie les règles d'association obtenues selon les catégories sur la base des conclusions. Lorsque la conclusion révèle une ou plusieurs mesures en relation avec la mesure pertinente, la performance de l'algorithme est évaluée; sinon, dans la mesure où elle est uniquement relative à la portée des éléments E, elle peut être utilisée pour accroître les connaissances à propos des caractéristiques d'un ensemble d'applications.

L'exemple ci-après démontre l'utilisation de la méthodologie : considérer un profil GAD E avec les paramètres suivants {20, 0.12, 0.56, 0.25, 0.13} et une mesure de performance *m* de valeur 2.66, m pouvant être le gain entre le temps d'exécution sur un processeur par rapport à l'exécution sur 3 processeurs. Supposer également que nous ayons R_N = {20,40,80} (un nombre de tâches dans une certaine plage.), R_C = {0.5,1,2}, R_σ = {0.5,1,2}, R_β = {0.1,0.5,1} et R_D = {0.25,0.5,0.75,1} pour E et R_m = {1, 2, 3} pour *m*. Alors, les éléments E possèdent aussi des représentations binaires, sous forme de zéros et de uns, R_N = 100, R_C = 100, R_α = 010, R_β = 010 et R_D = 0010, et la valeur de la mesure est codée R_m = 001. Par conséquent, la représentation binaire de la liste des attributs sera 100 100 010 010 0010 001. Si on note les bits successifs de cette séquence N1, N2, N3, C1, C2, C3, α1, α2, α3, β1, β2,

---

[1] L'algorithme fp-growth est un algorithme capable d'apprendre rapidement et de trouver tous les ensembles d'éléments fréquents dans un ensemble de données en utilisant une représentation extrêmement compressée des données de manière à se loger dans la mémoire principale, même pour les grandes bases de données. Il amorce sa recherche avec un support élevé et décroît de manière itérative de 20 % en tentant de trouver suffisamment d'ensembles d'éléments, environ 100 ou plus, pour générer des règles avec Rapid Miner.

β3, D1,D2,D3,D4,M1,M2,M3 en fonction des différents attributs qu'ils représentent et de leurs valeurs respectives, on peut aussi représenter E par la signature N1C1α2β2D3M3, signifiant qu'on y retrouve les attributs N1, C1, α2, β2, D3 et M3 avec les classes de valeurs spécifiées.

Avec la méthodologie décrite, l'expert peut considérer un groupe d'applications et utiliser les règles d'association obtenues pour modifier les paramètres d'un algorithme d'ordonnancement particulier ou en créer un nouveau. Dans la prochaine section, nous nous concentrerons sur une règle d'association ayant une mesure de performance comme conclusion.

3.2 Résultats expérimentaux

Nous démontrons ici comment l'apprentissage des règles d'association peut être effectué sur deux algorithmes d'ordonnancement nommés HEFT et CPOP, et permettre leur comparaison. Du point de vue de l'algorithme, l'hypothèse de la règle représente un GAD (caractéristique de l'application) et la conclusion est une mesure. Nous avons utilisé une mesure d'accélération pour les expériences, mais d'autres mesures peuvent être également utilisées (par exemple APD, SLR, contention, puissance, etc.). Nous avons déterminé un facteur d'accélération pour trois processeurs à 20 % ou plus afin de représenter un intérêt pour nous. Nous désirons savoir si l'un des algorithmes conduit à une valeur d'accélération supérieure ou égale à 20 % que nous indiquons par SUP et quand la valeur est inférieure à 20 % que nous indiquons !SUP. Supposons que M = {SUP, !SUP} donne les représentations logiques correspondantes, où !SUP est la négation de SUP. Le Tableau 3.1 indique certaines de ces règles d'association, extraites à partir de la base de données du test créée par l'utilisation du HEFT.

Comme l'indique le Tableau 3.1, la valeur Lift est plus grande que 1 et la valeur de confiance est élevée pour chacune des règles. Les règles 1 à 6 indiquent là où l'algorithme est fort en fonction du SUP et les règles 7 à 11 montrent où il est faible. En particulier, nous notons que HEFT obtient de faibles résultats lorsque α, le facteur de parallélisme, se trouve dans le domaine α1 (α1 représente une application avec un parallélisme faible, par exemple de 0 à

0.5). Le Tableau 3.1 montre que, dans ce cas (règle 8), 96.1 % des applications dans la base de données échouent à obtenir une accélération de 20 % lors de l'ordonnancement avec HEFT. Pour le profil de GAD décrit dans la section précédente, nous constatons que E est soumis aux règles 2, 3 et 6 et, par conséquent, nous nous attendons à un indice de confiance supérieur à 95 % tandis que HEFT l'ordonnancera avec une accélération de 20 %. En effet, l'accélération dans ce cas était de 2.66.

Tableau 3.1

Certaines règles HEFT extraites par l'utilisation de l'algorithme fp-growth

Règle	Prémisses	Conclusion	Support	Confiance	Lift
1	α 2, β3, C1	SUP	0.063	0.974	1.542
2	α2, C1	SUP	0.159	0.964	1.526
3	α 2, β2, C1	SUP	0.067	0.957	1.515
4	α2, C3	SUP	0.143	0.955	1.515
5	α 2, β2, C3	SUP	0.062	0.953	1.509
6	C1	SUP	0.175	0.799	1.264
7	α1,C4	!SUP	0.042	0.974	2.644
8	α1	!SUP	0.177	0.961	2.609
9	β2, α1	!SUP	0.077	0.960	2.607
10	B3, α1	!SUP	0.067	0.958	2.607
11	N1, α1	!SUP	0.043	0.876	2.380

(N(Nombre de noeuds),α(le taux de parallélisme), C(le rapport communication-calcul), β (la portée du calcul pour chaque nœud))

La règle 6 indique également que HEFT répond à l'exigence SUP 79.9 % du temps lorsque C1 seul est présent. Ce pourcentage grimpe à 96.4 % lorsque C1 et α2 sont tous deux présents (règle 2), α2 représentant un facteur de parallélisme entre 0.5 et 1.0.

Le Tableau 3.2 indique les règles extraites lors de l'utilisation du CPOP. Comme pour le Tableau 3.1, nous pouvons en tirer des conclusions. D'une part, les valeurs de confiance pour la force de l'algorithme sont inférieures par rapport au tableau 3.1. De même que pour HEFT, CPOP n'est pas approprié pour les applications avec $\alpha = \alpha_1$ (avec une valeur de confiance de

98.8 %). Sa valeur de confiance la plus élevée est 85.8 % et elle se produit pour l'hypothèse α2, β3, C1.

Tableau 3.2

Certaines règles CPOP extraites utilisant l'algorithme fp-growth

Règle	Prémisses	Conclusion	Support	Confiance	Lift
1	α_2, β_3, C_1	SUP	0.055	0.858	1.856
2	α_2, C_1	SUP	0.136	0.825	1.784
3	α_2, β_3, C_3	SUP	0.046	0.811	1.755
4	α_2, β_2, C_1	SUP	0.057	0.804	1.740
5	α_2, C_3	SUP	0.117	0.785	1.699
6	α_1	!SUP	0.182	0.989	1.838
7	α_1, β_1	!SUP	0.079	0.988	1.836
8	α_1, β_2	!SUP	0.069	0.988	1.836
9	N_1, α_1	!SUP	0.047	0.964	1.792

(N(Nombre de nœuds),α(le taux de parallélisme), C(le rapport communication-calcul), β (la portée du calcul pour chaque nœud))

Dans nos résultats, nous utilisons une accélération supérieure et inférieure à 20 % comme métrique de performance ; nous avons pu utiliser également l'ensemble échec, passable, bon et excellent comme domaine de valeurs. Les hypothèses et les conclusions subséquentes sont modifiées en conséquence.

Le Tableau 3.3 indique la façon dont nous pouvons exploiter les ensembles d'éléments pour comparer les algorithmes d'ordonnancement utilisant la même base de données d'application. Les règles marquées Δ représentent les mêmes règles d'association pour les deux algorithmes. Les quatre premières règles Δ représentent la force. et la cinquième règle: la faiblesse. Avec α_1 comme hypothèse, HEFT n'est meilleur que CPOP que de 2.8 %; néanmoins, avec α_2, C_3 l'écart s'accroît à 17 %. Il est également important de noter que CPOP est un ordonnanceur de chemin critique. Ainsi, de la même manière que pour les mesures APD et NB, notre approche est capable d'évaluer la dégradation globale ou le

nombre de solutions meilleures des deux algorithmes; elle offre aussi un meilleur aperçu et nous pouvons utiliser toutes les hypothèses et les conclusions jugées utiles.

Tableau 3.3

Comparaison de la confiance Δ des algorithmes HEFT et CPOP

Règle Δ	Règle HEFT	Règle CPOP	Prémisses	Conclusion	Δ confiance
1	1	1	α_2, β_3, C_1	SUP	11.6 %
2	2	2	α_2, C_1	SUP	13.9 %
3	3	4	α_2, β_2, C_1	SUP	15.3 %
4	4	5	α_2,C_3	SUP	17.0 %
5	8	6	α_1	!SUP	-2.8 %

(N(Nombre de noeuds),α(le taux de parallélisme), C(le rapport communication-calcul), β (la portée du calcul pour chaque nœud))

3.3 Complexité de la méthodologie

Cette méthode nécessite des informations sur les résultats de chaque algorithme d'ordonnancement, une fois extraits par l'outil d'évaluation à l'étape 1 de la figure 3.1. Pour extraire les règles, il existe plusieurs algorithmes. Certains algorithmes utilisent la génération de candidats [105], mais ils sont coûteux pour les grands ensembles de données et peuvent être exponentiels. Dans nos expérimentations, plus de cinquante mille profils GAD et mesures associées ont été générés, chacun avec une représentation binaire de 26 bits de longueur. Cela a conduit à plus de 1.5 million de possibilités examinées. Heureusement, Rapid Miner avec fp-growth ont généré les règles en moins de 15 s. En outre, fp-growth est efficace dans l'extraction des ensembles d'éléments fréquents sans génération de candidat ou en explorant les ensembles de données eux-mêmes. Cela permet au concepteur d'extraire une règle à la demande. Par exemple, si une règle d'association avec une hypothèse α3 C1 est souhaitée, les outils peuvent extraire cette règle particulière.

Les limitations principales de l'algorithme sont celles du fp-growth pour l'extraction des règles avec un nombre fixe d'itérations et la taille de la base de données de l'application. Aussi, l'exploration des données ne garantit pas de générer toutes les règles.

3.4 Conclusion

Nous avons proposé d'utiliser l'exploration de données pour extraire des connaissances sur des ensembles de données d'application et évaluer l'impact de l'algorithme d'ordonnancement sur les applications. Notre approche n'utilise pas de statistiques comme d'autres le font : elle analyse et extrait plutôt les connaissances à partir des données brutes. Nous avons montré que les règles extraites pouvaient être utiles de plusieurs manières afin d'évaluer les ensembles de données et la performance d'un algorithme d'ordonnancement. Nous avons aussi introduit le concept d'une Règle Δ pour comparer les algorithmes. Le but consistait à repérer les points forts et les faiblesses de chaque algorithme et à apporter ces informations à l'expert. À cet égard, la méthodologie d'introduction est flexible et exploite efficacement les règles afin d'accroître les connaissances sur les algorithmes d'ordonnancement.

Le prochain chapitre introduit une façon efficace de visualiser et comparer un ensemble d'algorithmes avec des règles d'association et de montre les possibilités de cette nouvelle approche lorsque des algorithmes multiples sont considérés.

CHAPITRE IV

CARTOGRAPHIE DE REGLES POUR LA CONNAISSANCE DES ALGORITHMES DE PLANIFICATION

Jusqu'à maintenant, plusieurs méthodes de représentation et de comparaison de données ont été proposées pour un ou deux algorithmes. Ce chapitre présente une approche pour la visualisation et la comparaison de plusieurs techniques d'ordonnancement. Dans cette section, nous proposons les cartes de règles comme outil d'analyse et de comparaison. Cette section présente du matériel qui a été publié dans un article paru dans les actes de IEEE *International Symposium on Circuits and Systems* (ISCAS' 2013) [106].

Dans ce chapitre, nous décrivons une approche de visualisation qui permet de comparer un nombre arbitraire d'algorithmes, en montrant les données pertinentes dans les différents points de vue, grâce à une carte de règles. Les cartes expriment les patrons d'attributs suivant une carte où des couleurs donnent une représentation visuelle des degrés de confiance des règles d'association correspondantes (analogie avec une carte thermique). Nous attribuons différents niveaux de couleurs selon les degrés de confiance pour chaque attribut et différents algorithmes ou métriques. Trois exemples de carte illustrent l'efficacité de la méthode avec 21 attributs. Le premier exemple illustre la comparaison de trois algorithmes. La deuxième carte montre comme utiliser l'approche avec un sous-ensemble de 15 sous-algorithmes d'un même algorithme. Cette carte permet de comparer les résultats de plusieurs sous-algorithmes d'un seul algorithme. La dernière carte montre comment nous pouvons visualiser l'impact de différentes métriques sur un algorithme. Nous décrivons une approche de visualisation qui permet de comparer un nombre arbitraire d'algorithmes, en montrant les données pertinentes sous des perspectives différentes grâce à une carte de règles.

Ce chapitre présente donc, dans un premier temps, notre méthodologie de cartographie pour extraire des informations sur les algorithmes de planification. Puis, nous exposons des

résultats expérimentaux montrant le potentiel des techniques de cartographie de règles que nous avons utilisées. Nous concluons sur la perspective qu'apporte cette méthodologie.

4.1 Méthodologie proposée

Prenons un DAG, dont nous voulons une image des performances lorsqu'une liste d'algorithmes d'ordonnancement s'y applique. Pour cela, nous proposons une approche visuelle inspirée des cartes thermiques où les couleurs remplacent les entrées d'une matrice. Soit une liste d'algorithmes LA, pour laquelle nous définissons une carte de règles en tant que matrice scalaire RM = $[R_{i,j}]_{1\leq i\leq m, 1\leq j\leq n}$ où chaque élément représente une information pertinente (généralement un indice de confiance) à propos d'un algorithme d'ordonnancement et d'une règle d'association. La carte de règles est ensuite convertie en une carte thermique en cartographiant ses éléments en gradients de zones de couleur grâce à un vecteur de cartographie que nous nommons GC. Dans cette section, nous montrons trois règles différentes permettant d'exploiter cette technique de visualisation : les règles, les règles Δ, les sous-algorithmes.

La première étape de construction de RM est la création de deux listes d'association de règles. La première est la liste de règles d'association d'algorithmes (AAR) contenant les règles d'associations pour un algorithme particulier A, appelé A.AAR. Il existe une liste comme celle-ci pour chaque algorithme d'ordonnancement. La deuxième liste est la liste de règle d'association (AR) qui est l'union de toutes les listes A.AAR. Les deux listes sont créées à partir du pseudocode suivant qui réalise l'union des ensembles:

1) Pour chaque algorithme d'ordonnancement A, lui créer un ensemble d'association de règle A.AAR pour une mesure de performance d'intérêt comme indiqué au chapitre 4.

2) À partir d'une liste vide AR, pour chaque algorithme A dans LA et chaque règle eR dans A.AAR : ajouter eR dans AR si eR n'en fait pas partie.

Par exemple, soit quatre algorithmes d'ordonnancement A, B, C et D avec l'ensemble de règles suivant :

A. AAR = {N1, N1C2, N1C3}

B. AAR = {N1, N1C2, N3C4}

C. AAR = {N1, N1C2}

D. AAR = {N1}

Nous avons AR = {N1, N1C2, N1C3, N3C4}.

Normalement, les listes AAR sont générées automatiquement au cours de l'étape 1) ci-dessus, mais il est aussi possible de créer une liste personnalisée nommée CAR, permettant de commencer avec une liste spécifique de règles d'association, par exemple CAR = {N1, N1C2}.

Une fois les listes AAR (ou CAR) et AR définies, il existe de nombreuses façons d'exploiter les cartes de règles et de créer RM. La plus évidente utilise l'indice de confiance de règle comme information à afficher. Le pseudocode suivant montre comme le créer avec AR ou CAR :

Cartes de règles courantes ou personnalisées

La première méthode utilise AR ou CAR en cinq étapes :

1. Choisir des règles de AR ou CAR pour définir la liste LR.

2. Pour chaque règle dans LR qui n'appartient à aucune liste AAR[2], appliquer l'équation de confiance pour déterminer son niveau d'indice de confiance pour les mesures de performances intéressantes et mettre à jour ses entrées de description.

3. Classer RM de manière à ce que l'antécédent de chaque règle dans LR soit l'abscisse

---

[2] Cela peut arriver avec une liste personnalisée ou un algorithme d'ordonnancement qui ne partage pas de règles avec celles qui ont déjà été étudiées.

i et que le nom de l'algorithme soit dans l'ordonnée j.

4. Associer chaque élément RM_{ij} à l'indice de confiance de règle correspondant.

5. Afficher chaque RM_{ij} en prenant du vecteur CG le gradient de couleur correspondant.

Le gradient de couleur correspond normalement à une plage de données pour l'indice de confiance d'une règle. Cela est utile pour montrer son niveau de sûreté en termes de patrons et d'antipatrons. Il s'agit d'identifier facilement des points dans RM qui indiquent quand un algorithme d'ordonnancement peut être appliqué en toute certitude ou non. Par exemple, la zone du bon patron suivra un dégradé de vert pour montrer les valeurs à fort indice de confiance, tandis que les zones antipatrons suivront un dégradé de rouge.

4.1.1 Règle des Cartes de règles Δ

Il est également possible d'utiliser une carte avec des règles Δ afin de mieux comprendre les performances des algorithmes en fonction d'une référence. Cette méthode est similaire à la précédente, sauf pour l'utilisation de l'indice de confiance Δ pour les éléments de RM à l'étape 4. Cela permet une identification aisée des algorithmes qui sont meilleurs ou pires qu'une référence, pour chaque ensemble de règles. Par exemple, on peut afficher une règle Δ meilleure de 15 % en vert et une règle Δ moins bonne de 15 % en rouge. Ainsi, lorsqu'on se trouve dans la zone verte, cela signifie qu'il vaut mieux garder l'algorithme de référence; ou bien, on peut améliorer la situation en utilisant les données de la zone rouge pour fusionner des algorithmes ou créer un algorithme de classe de règles comme dans l'article [108].

4.1.2 Carte de règles de sous-algorithmes

Nous définissons un sous-algorithme comme la même technique appliquée de manière différente. Une méthode d'ordonnancement comme HEFT possède plus de dix déclinaisons. On peut montrer l'impact que produit l'utilisation de différentes variations algorithmiques en créant et en visualisant les cartes de règles correspondantes. La procédure est la même qu'en première sous-section, mais nous utiliserons les noms des sous-algorithmes à la place de ceux des algorithmes. Chaque rangée représente alors une approche d'implémentation différente.

4.1.3 L'impact des estimateurs

Nous savons déjà que le choix des mesures et la manière dont nous les calculons ont un impact sur la détermination des performances d'un algorithme d'ordonnancement. Pour visualiser cette situation, nous utilisons une liste d'estimateurs de performance qui remplacent le nom des algorithmes. En prenant un algorithme intéressant, cela montrera l'impact de chaque mesure sur l'indice de confiance des différentes règles. Cette liste de règles peut provenir de AR ou de CAR et une colonne dotée de la même couleur indique que l'algorithme intéressant est cohérent avec les différents estimateurs.

4.2 Résultats expérimentaux

Nous avons utilisé l'approche de Topcuoglu et autres [13] pour générer des données à afficher grâce à notre technique de visualisation par carte thermique. La figure 4.1 illustre la comparaison de trois algorithmes d'ordonnancement, CPOP, PETS et HEFT, grâce à une carte thermique. L'axe x représente des valeurs attributs différentes provenant des signatures d'un groupe de DAG et l'axe y identifie ces algorithmes (cela aurait aussi fonctionné pour des valeurs de mesures de performances). La partie gauche de la figure 4.1 apporte l'indice de confiance de la règle pour la cartographie en couleur. Une zone rouge indique une règle avec un indice de confiance faible et une zone verte (près de 100) indique une règle avec un indice de confiance fort.

La figure 4.2 illustre la comparaison d'un ensemble de sous-algorithmes pour HEFT. Enfin, la figure 4.3 illustre l'impact de l'utilisation de différents estimateurs dans un algorithme d'ordonnancement. Dans cette figure, HEFT, HEFTB, HEFTW et HEFTME sont les coûts moyens, les meilleurs, les pires et médians utilisés par HEFT. Comme le montre chaque figure, l'utilisation d'une carte thermique facilite l'estimation de la performance d'un algorithme (ou l'efficacité d'une mesure de performance) en fonction de la présence d'une ou plusieurs valeurs attributs dans la signature du GAD.

Figure 4.1 : Exemple d'une carte de règles pour la comparaison d'algorithmes

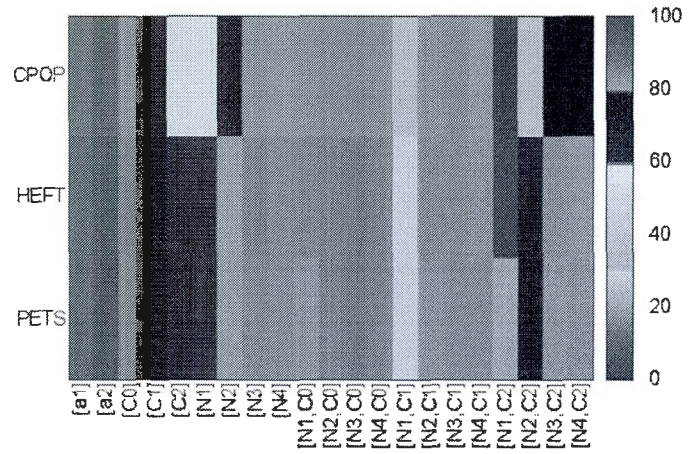


Figure 4.2 : Exemple de carte de règles pour la comparaison de sous-algorithmes

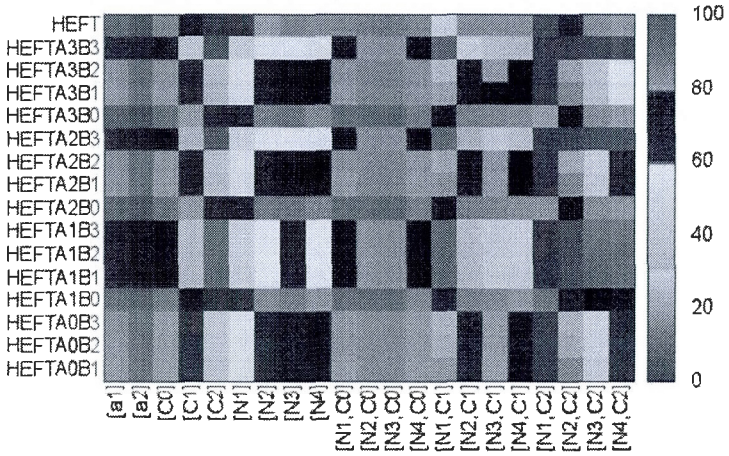
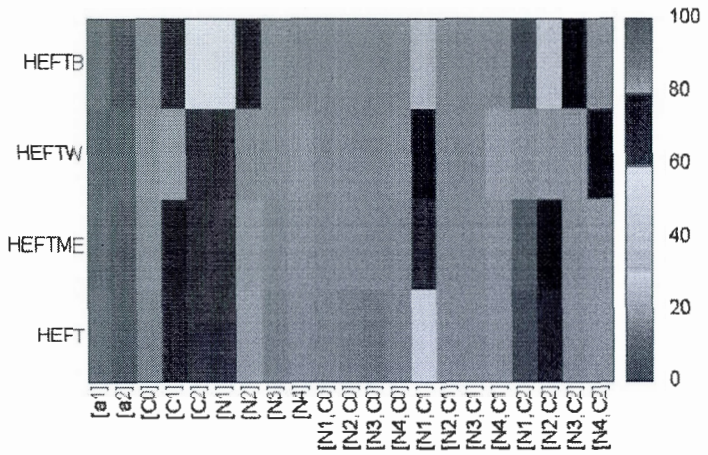


Figure 4.3 : Exemple de carte de règles pour la comparaison des effets des paramètres



Nous pouvons remarquer que les règles d'association de ces cartes nous permettent d'obtenir des granularités différentes des informations, les valeurs de l'axe x pouvant représenter des valeurs attributs simples ou multiples. Par exemple, la méthodologie peut n'utiliser que deux attributs importants, N et CCR, et le concepteur peut séparer un ensemble d'applications en n groupes qui sont représentés par les colonnes de la carte. C'est une manière de représenter les données de performance plus efficace que l'utilisation d'un format texte.

4.3 Conclusion

Nous avons exposé une technique de visualisation de cartes de règles afin de comparer des algorithmes d'ordonnancement et apporter des exemples d'utilisation. La méthodologie est applicable à un algorithme unique ou à un groupe et elle permet notamment d'identifier facilement des points posant problème sur la carte de règles. L'axe y sur la carte thermique peut représenter des algorithmes ou des mesures pour évaluer leurs performances, individuellement ou en groupes. Par conséquent, l'interprétation de la carte thermique dépend du type de données utilisé dans la carte de règles.

L'utilisation d'une carte thermique apporte au concepteur une connaissance accrue des différents algorithmes d'ordonnancement et les combinaisons possibles des variables des axes x et y sont presque infinies. En conclusion, l'utilisation de l'apprentissage d'associations

de règles pour en tirer des cartes de règles et la méthode de visualisation décrite dans ce chapitre apporte une perspective nouvelle sur les algorithmes d'ordonnancement; cela peut être un complément utile aux références traditionnelles.

Jusqu'à maintenant, nous avons utilisé les règles d'associations avec plusieurs nouvelles techniques en vue de comparer et d'exploiter les nouvelles connaissances. Bien que ce soit efficace en termes de règles d'association, nous devons trouver une façon de représenter et manipuler les connaissances algorithmiques sous forme de collection. Le prochain chapitre ajoute la notion des langages-patrons pour représenter plus efficacement encore les algorithmes d'ordonnancement.

CHAPITRE V

LANGAGE DE PATRONS POUR REPRESENTER LES CONNAISSANCES DES ALGORITHMES D'ORDONNANCEMENT

Dans le chapitre II, nous avons introduit l'usage de langage de patrons.

Dans l'optique de représentation des connaissances hétérogènes sur les ordonnanceurs statiques pour aider notre concepteur d'applications à faire son choix selon son architecture ciblée, nous allons utiliser les patrons pour pouvoir encapsuler les connaissances des algorithmes statiques, car ceux-ci permettent de faire des catalogues de solutions. De plus, chaque patron ou ensemble de patrons peut être modélisé avec un langage de modélisation de patrons et une grammaire.

Dans cette perspective, nous montrons l'utilité et la flexibilité des langages de patrons et leurs grammaires et comment utiliser ceux-ci pour saisir et manipuler les connaissances au sujet des algorithmes d'ordonnancement. Nous montrons comment à partir des connaissances disponibles comment les représenter sous forme de patrons. Nous utilisons des algorithmes de quatre classes : liste, aléatoire, duplication et partition. Nous utilisons des applications comme la FFT et la factorisation Cholesky.

Ce chapitre débute par la méthodologie que nous proposons, il est suivi de l'exposé de l'environnement de connaissance, des résultats que cela génère et d'une discussion portant sur ceux-ci. Nous concluons sur les impacts d'une telle approche.

5.1 Méthodologie proposée

Dans cette section, nous nous concentrons sur les modèles et étendons certains concepts et résultats que nous avons proposés aux chapitres II et III. Nous discutons des langages de patrons avec les règles d'association et de la grammaire. La section suivante détaille nos définitions de patrons et de grammaire.

5.1.1 Langage de patrons pour l'ordonnancement et les patrons connexes

Nous définissons un langage de modèle d'ordonnancement en utilisant une définition syntaxique formelle [95]. Un DAG SPL = <SP, SR> où SP = $\{SP_1 ... SP_n\}$ est un ensemble de nœuds qui représentent des patrons sur des algorithmes d'ordonnancement et où SR = $\{SR_1 ... SR_M\}$ est un ensemble d'arêtes dirigées entre les nœuds(patrons). L'ensemble des arêtes sortant d'un nœud est appelé références et l'ensemble des arêtes qui entrent d'un nœud est nommé contexte. Avec cette définition, un algorithme d'ordonnancement est considéré comme un sous-ensemble de SPL avec des éléments de SP et de référence des liens de SR, et où les nœuds sont tels que :

- Chaque élément $sp \in SP$ est appelé un patron.
- Pour chaque patron sp dans SP, $Q \in SP$, sp références $Q \Leftrightarrow \exists\ sr = (sp, Q) \in SR$.
- Chaque nœud représente un patron.

Un patron d'ordonnanceur se compose d'un n-uplet incluant un nom *n*, un contexte *c*, un classement *r*, une illustration *i*, un problème *p* avec les forces $f_1 ... f_i$, exemples $e_1 ... e_j$, une solution *s* et un schéma *d*.

Plus précisément :

- Le nom du patron fait rapidement référence à une situation donnée, une solution.
- Le contexte résume la performance d'un patron – par exemple soit le temps d'exécution ou sa consommation d'énergie - et l'implémentation de l'architecture(s)

où il est pertinent – par exemple, trois processeurs ou plus, connectés pleinement–. L'information est définie par deux ensembles : CapabilitySet et Tech. En bref, le champ contextuel indique les cas de conception où le modèle peut être utilisé.

- Le classement est utilisé dans le sens de Liou et Palis [95][3]. Il exprime notre confiance sur l'applicabilité du modèle à un algorithme d'ordonnancement. Dans notre environnement, nous utilisons le niveau des règles d'association obtenues entre un DAG, la signature ou des éléments d'une application de celui-ci et les résultats de performance pertinente comme métrique pour la fiabilité et la confiance du patron. Cette information est stockée dans deux ensembles, FSET et WSET, qui énumèrent les signatures avec des niveaux de confiance forts et bas respectivement pour les règles d'association. Différents seuils et des conditions peuvent être utilisés pour plus de polyvalence.

- L'illustration identifie une situation typique d'application de cette solution. Le problème identifie les applications dans lesquelles le modèle s'applique. Le problème d'ordonnancement est décrit en termes de règles d'association entre les propriétés de l'application et les résultats de performance. Normalement, la prémisse de chaque règle d'association est la signature d'un DAG ou d'une partie. Ensuite, nous mettons dans FSET et WSET les signatures d'application DAG, qui représentent les ensembles des forces et des faiblesses des applications selon notre classement.

- Des exemples sont des situations qui peuvent nécessiter le modèle et la façon dont le modèle peut résoudre le problème. Des mesures, métriques, pourraient être également possibles.

- La solution identifie une solution éprouvée au problème basé sur une généralisation à partir des exemples qui équilibrent les forces. Solutions : généraliser à partir

---

[3] Tang et autres [17] définissent le classement comme "Le classement montre comment universellement valide l'auteur du patron croit en son patron. Il aide le lecteur à distinguer les premières idées des modèles qui ont été confirmés à maintes reprises."

d'exemples éprouvés pour équilibrer les forces et apporter des solutions à un problème. Ceci est l'algorithme de planification, notre solution sous forme de patron, dans notre cas. Nous définissons l'ISET {L, BNiveau, CP}, qui fournit des propriétés intrinsèques de l'algorithme. L est un ordonnancement de liste (une classe d'algorithme). BNiveau indique que l'algorithme utilise cette métrique de calcul et le CP considère le chemin critique.

- Schema soutenant la solution de manière graphique.

Par définition, un patron fait usage d'un certain nombre de sémantiques qui permettent de capturer un problème de design récurrent et suggèrent une solution éprouvée pour elle.

5.1.2 Grammaire

En dehors d'un langage de patrons, nous avons besoin d'une grammaire pour manipuler les connaissances. Nous définissons notre grammaire selon Zdun [91]. D'où une grammaire non contextuelle G d'un langage SPL est définie comme le n-uplet G = (V, T, P, S) où :

- V est un ensemble de variables syntaxiques (non-terminaux).
- T est un ensemble de symboles terminaux qui sont un sous-ensemble de l'alphabet A du langage L. Dans notre cas, les symboles de fermeture peuvent être des modèles, des alternatives et des architectures.
- P est un ensemble de règles de production $X \rightarrow Y$, ce qui signifie que X peut être transformé en Y. X et Y peuvent contenir des variables syntaxiques et des symboles terminaux.
- S est un symbole de départ, $S \in V$, qui commence l'application de règles de production. Nous utilisons le terminal **Départ**.

En utilisant la notation Backus Naur Form (BNF) pour notre grammaire, nous obtenons:

Identifiant ::= Lettre { Lettre| Chiffre}
Signature ::= Identifiant
Approche ::= Identifiant «;» [Options] «;» Requis [**FSET** FSET] [**WSET** WSET] [**Classement** Classement] [**Tech** Tech] [**CapabilitySet** CapabilitySet] [**ISET** ISET]
Départ ::= **Départ** Approche
Architecture ::= Identifiant
Requis ::= Architecture
Options ::= Approche {«,» Approche}
Production ::= Variables « → » Approche | Options | Requis | Architecture | Identifiant | Signature
Variables ::= Approche | Options | Départ | Requis | Architecture | FSET | WSET | Classement
FSET ::= Signature {«,» Signature }
WSET ::= Signature {«,» Signature }
Tech ::= Architecture {«,» Architecture }
Classement ::= Identifiant
Application ::= Identifiant ; Signature ; Requis ; Classement
Capability ::= Identifiant
CapabilitySet ::= Capability {«,» Capability }
ISET ::= Identifiant {«,» Identifiant }

L'identifiant est une représentation de lettres et de chiffres. L'approche consiste à un identifiant, des options et un requis (une architecture ciblée). La sélection initiale Départ permet de démarrer l'application des règles. Les règles de production produisent une

approche, des options, un requis. Nous arrêtons la production de règles lorsque tous les éléments sont des identifiants. L'approche contient les éléments importants que nous utilisons tels que FSET, WSET, etc. Les accolades indiquent les champs qui peuvent être omis.

Nous utilisons cette grammaire pour manipuler les patrons des algorithmes de planification, la recherche d'options alternatives, etc. avec la définition de l'application de l'usager. La grammaire crée également une couche d'abstraction supérieure pour la représentation des algorithmes d'ordonnancement et applications où les opérations grammaticales sont génériques. La prochaine section montre un exemple simple d'application de la grammaire avec des séquences de patrons produits par les règles de production comme décrits par Zdun [91].

5.1.3 Exemples : grammaire et séquences

Par exemple, étant donnée une approche algorithmique nommée *AlgoA* permettant l'ordonnancement sur une architecture à trois processeurs nommé *ArchA* avec des options d'ordonnancement *AlgoB* et *AlgoC*. Nous utilisons la grammaire suivante:

```
Arch1 = «Architecture3Processeurs»
Algo2 = «AlgoB » ; Arch1
Algo3 = «AlgoC » ; Arch1
Algo1 = «AlgoA »; {Algo2, Algo3}; Arch1
Départ Algo1
```

À noter que les signatures des GADs permettent d'avoir les options compatibles selon notre approche, des patrons compatibles. Les options représentent également des liens vers d'autres patrons.

Pour obtenir une séquence, une liste de chaînes d'options, avec les règles de production, nous parcourons les approches ainsi que les options et le requis.

Nous démarrons avec Départ Algo1 :

1. Départ Algo1
2. «AlgoA» Algo2 Algo3 Arch1
3. «AlgoA» «AlgoB» «AlgoC» «Architecture3Processeurs »

La ligne 3 contient alors seulement des identifiants avec une architecture cible et une liste d'algorithmes compatibles.

Nous pouvons filtrer la séquence obtenue dans la séquence précédente pour n'obtenir que des informations d'intérêt. En effet, la grammaire permet l'extraction de tout ou d'une partie seulement des éléments : les options, les variables, etc. Par exemple, nous pouvons ignorer les exigences et les options des balises lors de l'exécution de la grammaire et obtenir :

1. Départ Algo1
2. «AlgoA» Algo2 Algo3
3. «AlgoA» «AlgoB» «AlgoC»

La séquence représente un ensemble d'algorithmes de planification qui sont soit des solutions soit des options de la même séquence.

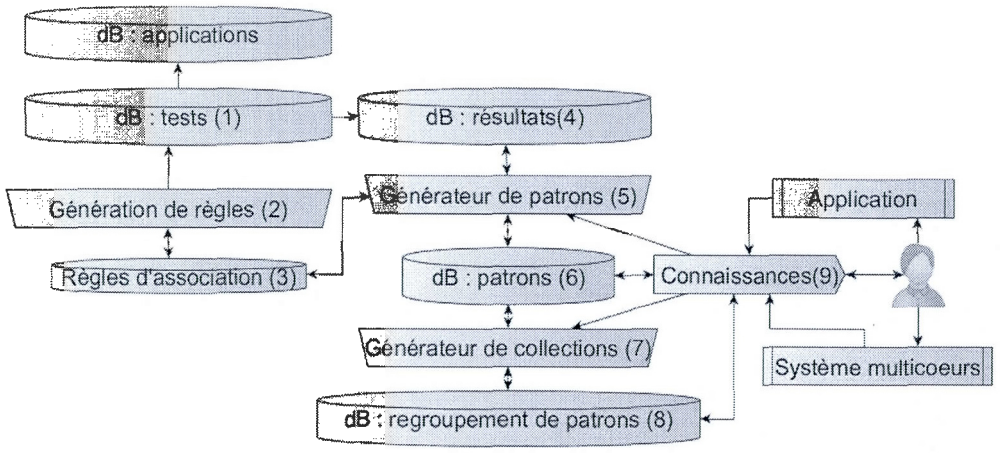
5.2 Environnement de connaissance

5.2.1 Vue d'ensemble

L'usage des règles d'association, d'un langage de patrons d'ordonnanceur et d'une grammaire qui permet la création efficace d'un environnement de gestion de connaissance pour l'ordonnancement comme le montre la figure 5.1 (où les littéraux et les numéros de

blocs sont utilisés pour identifier les différents composants). Il y a deux blocs fonctionnels principaux : le bloc de connaissances (bloc 9) et le générateur de patrons (bloc 5). Le bloc de connaissance est l'interface entre un usager-concepteur ou un outil – qui a une application à ordonnancer sans être expert sur les algorithmes d'ordonnancement. Les connaissances patrons et les techniques de visualisation sont utilisées pour donner à l'usager des renseignements à propos des algorithmes d'ordonnancement. Le second bloc important est le générateur de patrons (bloc 5) qui construit des modèles personnalisés à partir d'informations recueillies auprès de différentes sources et permet la modélisation des algorithmes d'ordonnancement.

Figure 5.1 : Environnement de connaissance de patrons



Les autres blocs sont : bloc 1 qui teste les algorithmes de planification pour la performance et met les résultats dans le bloc 4. Le bloc 2 extrait les informations du banc de test pour créer et gérer les règles d'association du bloc 3, et ce dernier extrait les données pour le générateur de patrons du banc d'essai du bloc 1. Le générateur de patron peut exploiter les données brutes de connaissances entreposées et/ou les règles d'association.

Le générateur de collection de patrons (bloc 7) étend le générateur de patron en groupant et, éventuellement, en affichant des patrons de signature de GAD par rapport à un contexte

d'application. Par exemple, il est possible de regrouper tous les algorithmes qui ordonnancent sur trois processeurs pleinement connectés comme une collection. En somme, le générateur de modèle génère des patrons à partir des algorithmes et le générateur de collections groupe ceux-ci.

5.2.2 Usage d'un patron

Comme mentionné plus tôt, un patron d'ordonnancement consiste en un n-uplet qui identifie les propriétés d'un algorithme d'ordonnancement et met en évidence son champ d'applications. Plus de détails sur les champs importants de l'environnement sont décrits :

- Nom : fait référence à une technique d'ordonnancement unique, comme HEFTA3B2. Dans ce cas, HEFT représente un algorithme principal et A3B2 indique une variante. Chaque sous-algorithme peut avoir un nom distinct et est considéré comme un algorithme.
- Contexte : contient les considérations pertinentes au sujet de l'utilisation du patron, typiquement une architecture cible et une attente sur les performances (les patrons d'architectures ne sont pas le sujet de ce travail). Ceci inclut des variables requises dans la grammaire.
- Classement : spécifie le niveau de confiance du patron quand les signatures GAD contenues dans FSET (ensemble des forces) et WSET (ensemble des faiblesses) identifient les situations où l'algorithme d'ordonnancement performe bien ou moins bien. WSET joue un rôle opposé en indiquant quand le patron n'est pas bon pour l'application.
- Le classement d'un patron peut être réalisé avec le forage de données sur l'ordonnancement comme présenté à la figure 5.1, blocs 1 à 3.

5.2.3 Sélection d'algorithme d'ordonnancement

Il y a deux principales méthodes pour fournir des patrons de technique à l'usager. La première consiste à balayer la base de données des patrons (bloc 6 de figure 5.1) pour un

match d'ensemble; la seconde consiste à créer un patron basé sur les données de l'usager. Ces méthodes vont être présentées dans les deux prochaines sous-sections.

a) Ensembles assortis

Nous commençons par définir un ensemble U en utilisant notre grammaire avec une application:

```
Tech ::= Architecture {«,» Architecture }
Application ::= Identifiant ; Signature ; Requis
```

Ceci résume la demande de l'utilisateur en deux sous-ensembles U.DAG (l'application) et U.TECH(Tech). U.DAG contient la signature de la demande et U.TECH contient le type d'architecture que l'utilisateur souhaite, le requis. Ainsi, U spécifie l'application, l'architecture cible et les indicateurs de performance choisis. Cette information est ensuite convertie en règles d'association.

Appelons SP, une approche, un ensemble composé de sous-ensembles TECH, FSET, WSET et CapabilitySet, où les sous-ensembles sont tels que définis précédemment. Nous voulons un ensemble de modèles SS qui peut résoudre le problème de l'utilisateur tout en répondant aux exigences :

- $SS \in SPL$
- $U.TECH \cap SP.TECH \neq \{\}$
- $(U.DAG \cap SP.FSET) \setminus SP.WSET \neq \{\}$

Cependant, cette méthode peut fournir $SS = \{\}$. L'autre solution consiste à générer un modèle personnalisé avec des données fournies par l'utilisateur.

b) Génération de modèle personnalisé

Cette procédure peut être utilisée pour enrichir la base de données avec les connaissances patrons. Les informations nécessaires pour acquérir de nouvelles connaissances sont l'algorithme lui-même et le type d'optimisation qui peut être fait, CapabilitySet et la liste de compatibilité de Tech. Nous avons aussi besoin de récupérer les indicateurs de performance

souhaités. Tech peut être défini manuellement ou affecté lorsque nous testons l'algorithme sur une nouvelle configuration automatiquement.

L'aspect important est que le champ problème dans un patron représente des antécédents de règles illustrant un ensemble d'expériences GAD.

La conséquence de l'association est un ensemble d'attributs binaires qui représente les métriques. Il comporte trois étapes :

1. Construire la liste des prémisses voulues (appelons-la PL), les métriques utilisées et tous les paramètres pour valider le classement. Par exemple, nous allons utiliser SUP.
2. Retirer des PL toutes les prémisses qui ne se qualifient pas dans le classement.
3. Compacter les attributs. Ceci fournira des informations-raccourci pour l'utilisateur.

Il existe différents cas dont il faut tenir compte pour une prémisse. Celle-ci peut être filtrée avec certains paramètres tels que la valeur du lift ou d'un minimum de données. Ces considérations peuvent être énoncées dans l'élément de classement.

5.2.4 Exemples : Grammaire et Environnement

Cette section comporte quelques exemples sur l'environnement et la grammaire.

Soit un patron défini pour une méthode X concernant des applications ayant de 1 à 20 nœuds et 21 à 40 nœuds, respectivement, et pour des situations où les algorithmes donnent des gains de vitesse de 20% avec un niveau de confiance de 80%. Étiquetons les nœuds requis N_1 et N_2 avec la performance algorithmique SUP pour simplifier. Le générateur de patrons commence par chercher au bloc 3 la liste des règles d'association compatibles. S'il n'y en a pas, il vérifie dans le bloc 4 pour des résultats entreposés de GAD qui ont un gain de 20% avec l'algorithme X. S'il n'y a rien, il peut aller au bloc 1 et commencer à générer des résultats à partir de banc de tests. À la fin, le générateur de patron met N1 ou/et N2 dans FSET du patron de l'algorithme X s'il y a assez de GADs ou l'algorithme X rencontre le classement spécifié pour SUP. Après cela, le générateur de patrons met le patron dans le bloc 6. De plus,

supposer que l'algorithme X a FSET= $\{N_1\}$ et algorithme Y a le même FSET, le générateur de collection de patrons peut regrouper X et Y dans un concept parce qu'ils partagent N_1.

Le classement est un élément important de notre approche car elle est basée sur les règles d'association et peut manipuler celle-ci par les signatures. Par exemple, supposons que le champ de contexte stipule 20 % en gain d'accélération, mais le classement spécifie 80 % de niveau de confiance pour les signatures : N_0C_0, N_1C_0, et N_2C_0 et seulement 3 % pour la signature GAD N_3C_0. Nous pouvons écrire FSET= $\{N_0C_0, N_1C_0, N_2C_0\}$ et WSET= $\{N_3C_0\}$, ou alternativement FSET=$\{C_0\}$ et WSET=$\{N_3C_0\}$. Ceci veut dire que quand C_0 est présent dans la signature GAD, le patron a une bonne perspective pour le cas N_3C_0.
Par exemple, $N_1C_2 \Rightarrow$ SUP2 pourrait indiquer qu'un algorithme a une accélération de 40 % lorsqu'il est appliqué à GAD signature N_1C_2. FSET est construit à partir des règles d'association (3) à la figure 5.1.

Pour construire le classement avec FSET et WSET, nous pouvons utiliser la méthode de modèle personnalisé.

1. Supposons N_1 et C_1 à C_3 avec une prémisse de ces règles : $\{N_1$ (80), N_1C_1 (85), N_1C_2 (90) et N_1C_3 (40)$\}$. 80, 85, 90 et 40 représentent le niveau de confiance. Nous pouvons écrire FSET $\{N_1\}$ et $\{$WSET $N_1C_3\}$ ou $\{$FSET N_1C_1, N_1C_2, et $N_1C_3\}$. Ceci permet une flexibilité de représentation et de visualisation des modèles pour l'utilisateur.

2. WSET est construit de la même manière que FSET, mais il utilise de faibles niveaux de confiance. Un exemple de la génération d'un modèle pour un algorithme d'ordonnancement suit : soit un algorithme A du type de la classe d'algorithme liste et a été testé sur trois processeurs connectés dans un système hétérogène (donc TECH = {3FCP}, où 3FCP résume l'architecture). Nous utilisons une banque de signatures applications dans la liste $\{N_1, N_2, N_1C_0, N_2C_1, C_0, C_1\}$ à partir desquelles, avec 80 % de confiance, certaines applications auront une vitesse d'au moins 20 % lors de l'application de l'algorithme tandis que d'autres connaîtront un ralentissement.

L'algorithme utilise le calcul du coût moyen (MA métrique) et l'accélération, d'au moins 20 % avec un niveau de confiance de 80 %, est représentée par un attribut binaire SUP. Cela conduit à la définition suivante :

Nom : Algorithme A

Contexte :	Système hétérogène avec trois processeurs et une architecture pleinement connectée; un gain d'accélération de 20 % et plus.
Classement :	FSET niveau de confiance ≥ 80 %; WSET niveau de confiance ≤ 20 %; gain d'accélération ≥ 20 % (SUP = vrai) quand l'estimateur est MA.
Problème :	FSET= $\{N_2, C_0\}$, ceci veut dire que N_2 et C_0 ont le classement de 20 % de gain d'accélération avec un niveau de confiance de 80 %. WSET= $\{C_1\}$, ceci veut dire que C_1 a un classement de moins de 20 % de gain d'accélération avec un niveau de confiance de 80 %.
Solution :	Technique d'ordonnancement algorithmique. ISET={L} (L pour type d'ordonnancement par liste)
Exemples :	Applications qui rencontrent le classement du patron.

Pour un utilisateur qui souhaite ordonnancer une application avec U.DAG = N_1C_2 et U.TECH = 3FCP (ce qui signifie un déploiement sur trois processeurs connectés), avec une accélération du temps d'exécution d'au moins 40 %, les règles d'association seront $N_1C_2 \Rightarrow$ SUP2 où SUP2 représente la métrique pour une accélération de 40 %. La grammaire suivante peut être utilisée.

```
Requis = «3FCP»
Classement  = SUP2
U= «MonApplication» ; N2C1 ; Requis ; Classement
```

Nous pourrions extraire des patrons qui peuvent répondre à la spécification d'un niveau de 80 % de confiance fixé par un expert ou l'utilisateur.

Dans la section suivante, nous montrons des résultats différents de l'application de nos outils.

5.3 Résultats : patrons pour les algorithmes d'ordonnancement

Cette section montre comment appliquer notre méthodologie et la grammaire à cinq algorithmes. La première sous-section porte sur la modélisation des méthodes de planification HEFT, PETS, CPOP, HEFTA1B1 et HEFTA2B2; les deux derniers sont deux variations de HEFT. La première étape consiste à entrer des informations sur les méthodes. Nous montrons des informations contenues dans les modèles, avec des commentaires explicatifs au besoin en italiques.

5.3.1 Patrons

Les trois premiers patrons sont décrits ici :

Nom : HEFT (P_1)

Contexte :	Système hétérogène avec plusieurs processeurs et architecture entièrement connectée TECH : {3FCP} *Gain de vitesse de 20 %* CapabilitySet = {SUP}
Classement :	Configuration : Confiance > = 80 % pour FSET et < = 20 % pour WSET, gain de 20 %. Estimateurs utilisés : MA
Problème :	FSET = {N_3, N_4, C_0, N_2C_1, N_3C_1, N_4C_1, N_3C_2, N_4C_2} WSET = {N_1C_2}
Solution :	L'algorithme d'ordonnancement ISET = {L, RU} *(L pour type d'ordonnanceur par liste et RU indique qu'il utilise une méthode de priorité ascendante)*
Exemples :	Banc d'essai filtré et application des connaissances d'applications passée

Nom : PETS (P_2)

Contexte :	Même que le patron HEFT
Classement :	Même que le patron HEFT
Problème :	FSET = {N_3, N_4, C_0, N_2C_1, N_3C_1, N_4C_1, N_3C_2, N_4C_2} WSET = {}
Solution :	L'algorithme d'ordonnancement ISET = {L, DRC, DTC, ACC}*(DRC, DTC et ACC sont trois formules qu'utilise l'algorithme)*

Nom : CPOP (P_3)

Contexte : Même que le patron HEFT
Classement : Même que le patron HEFT
Problème : FSET = {N_3, N_4, C_0, N_2C_1, N_3C_1, N_4C_1}
WSET = {N_1C_2}
Solution : L'algorithme d'ordonnancement ISET {L, CP, RU} *CP spécifie que l'algorithme utilise une approche par chemin critique.*

Les deux patrons suivants correspondent à deux variantes de l'algorithme HEFT :

Nom : HEFTA1B1(P_4)

Contexte : Même que le patron HEFT
Classement : Même que le patron HEFT
Problème : FSET = {N_2C_0, N_3C_0}
WSET = {N_1C_2}
Solution : L'algorithme d'ordonnancement ISET {L, RU, TA1, TB1}*(TA1 et TB1 représentent différentes politiques utilisées)*

Nom : HEFTA2B2 (P_5)

Contexte : Même que le patron HEFT
Classement : Même que le patron HEFT
Problème : FSET = {C_0, N_3C_1}
WSET = {}
Solution : L'algorithme d'ordonnancement ISET {L, RU, TA2, TB2} *(TA2 et TB2 indiquent des politiques utilisées)*

Dans tous ces modèles, il importe de savoir que lorsqu'une composante de la signature DAG est présente dans FSET – dire que nous avons C_0 dans la liste des attributs de HEFTA2B2 –, nous savons que l'algorithme d'ordonnancement répond à l'exigence du classement pour l'application. Cependant, l'absence d'un tel composant – dire que nous n'avons pas N_2C_0 dans FSET pour HEFTA1B1 – ne signifie pas nécessairement que nous nous retrouvons devant un mauvais choix, mais seulement que nous ne savons pas. Le mauvais classement ne peut être utilisé que si la composante a été spécifiquement incluse dans WSET. Ceci dit, ISET est également utile si nous voulons chercher des propriétés intrinsèques d'un algorithme.

5.3.2 Construction de séquences

Cette section illustre la construction de séquences de patrons pour répondre à toutes les options et les exigences appropriées à une application donnée. Cet exemple consiste en un graphique DAG avec N2C0 dans la signature et HEFT comme algorithme d'ordonnancement. Nous commençons avec une grammaire contenant les éléments suivants :

```
Requis = «3FCP»
P3 = «CPOP» ; Requis
P2 = «PETS» ; Requis
P4 = «HEFTA1B1» ; Requis
P5 = «HEFTA2B2» ; Requis
P1  = «HEFT» ; {P2, P3, P4, P5}; Requis
Départ P1
```

Ensuite, en commençant par P1 et en appliquant les règles de production sans le requis, la séquence suivante peut être obtenue :

1. P1
2. «HEFT» P2 P3 P4 P5
3. «HEFT» «PETS» «CPOP» «HEFTA1B1» «HEFTA2B2»

La séquence est finale avec les terminaux est :
«HEFT» «PETS» «CPOP» «HEFTA1B1» «HEFTA2B2».
Ceci permet à l'utilisateur de sélectionner l'algorithme d'ordonnancement avec une liste de variations compatibles et les options disponibles, dont les algorithmes d'ordonnancement qui s'y appliquent, étant donné le classement de performance sélectionné. Il existe un grand nombre de possibilités dans notre environnement pour améliorer la représentation et la manipulation des connaissances. La décision d'utiliser un algorithme ou une autre peut être capturée par le modèle choisi. En outre, nous pouvons facilement développer des alternatives et des options de perspective et de production de règles d'une règle d'association.

Les deux prochaines sous sections présentent des résultats concernant l'exploitation des modèles en termes d'algorithme d'ordonnancement par rapport à nos cinq patrons. La première partie utilise une approche de graphes d'applications aléatoires et, pour la sélection d'algorithme, des modèles décrits dans la dernière section. La seconde section montre notre méthode avec une structure de GAD du type utilisé pour le calcul parallèle d'un papillon dans une transformée de Fourier rapide (FFT), avec différentes valeurs de CCR.

Applications aléatoires

Notre méthode permet de sélectionner les algorithmes en fonction de leur motif au lieu d'un algorithme générique. Pour illustrer notre propos, nous avons utilisé dix applications graphiques aléatoires avec des paramètres différents. Chaque modèle P_1 à P_5 comporte des forces et faiblesses. Le tableau suivant 5.1 montre la métrique d'accélération pour chaque algorithme, liée à cinq modèles. Le symbole * indique que le patron correspondant répond au niveau d'une accélération de 20 % requis comme entendu de chaque règle d'association et de niveau de confiance entre le patron et la signature GAD (précédemment établie dans une expérience séparée) et, par conséquent, peut être une solution potentielle.

Le tableau montre que la signature N_1C_2 constitue une faiblesse pour P_1 à P_4 et n'est pas la force de P_5 non plus. En effet, ces modèles ne sont pas de bons candidats. Inversement, les modèles P_1 à P_3 sont de bons candidats pour N_4C_2; en effet, N_4 est dans FSET pour ces motifs.

Tableau 5.1

Applications aléatoires

DAG signatures	P_1	P_2	P_3	P_4	P_5
N_0C_0	72*	68*	74*	57*	23*
N_0C_1	32*	24*	13	-15	10
N_1C_1	55*	95*	51*	-4	45*
N_3C_1	162*	144*	106*	20*	84*
N_4C_1	174*	197*	104*	15	87*

N_1C_2	-17	-11	-32	-36	-53
N_1C_2	-24	-14	-26	-60	-57
N_4C_2	70*	100*	61*	-19	22
N_4C_0	189*	344*	112*	72*	97*
N_3C_1	138*	252*	101*	27*	59*

Applications particulières

Cette section traite le cas où une application spécifique d'un usager est d'ordonnancer un GAD avec trois différentes valeurs de CCR, avec trois designs possibles comme résultat. Ici, le but est de trouver un algorithme prometteur pour l'ordonnancement de trois GAD structurés pour la même application. Le tableau 6.2 montre une accélération différente pour chacun des cinq patrons algorithmiques, lorsque nous les appliquons à nos trois structures de GAD. Il semble que les signatures N0C1 et N0C2 n'ont pas déclenché de patron parce qu'elles ne sont pas dans le FSET. Même chose concernant P4 avec la signature N0C0; d'un autre côté, P_1, P_2, P_3 et P_5 sont candidats pour le classement, parce que les algorithmes correspondants donnent un gain d'accélération de 20 %.

Tableau 5.2

Application avec différentes valeurs CCR

DAG signatures	**P_1**	**P_2**	**P_3**	**P_4**	**P_5**
N_0C_0	117*	177*	119*	12	59*
N_0C_2	-9	14	8	-33	-50
N_1C_0	-29	-22	-9	-39	-11

(Chiffres en %)

Les chiffres positifs indiquent le gain du temps d'exécution et les chiffres négatifs signifient une perte du temps d'exécution par rapport à la référence, sur un seul processeur.

5.3.3 Applications diverses avec algorithmes de divers types

À partir de différents algorithmes d'ordonnancement et de données disponibles, nous allons montrer que notre approche est non seulement bon pour les algorithmes de type liste, mais bon pour les méthodes basées sur la recherche aléatoire, les partitions et la duplication. Vous allez comprendre que l'usage de patron dans le contexte de l'ordonnancement est une pratique nécessaire afin de conserver et d'expliquer les choix algorithmiques et architecturaux. Ainsi, un concepteur dispose des informations nécessaires pour justifier ses choix. Notre méthode permet dans un environnement de design de complémenter et d'expliquer les choix entrepris. Prenons des algorithmes de 4 classes : liste(CPOP,MCP,ETF), partition(MD,DCP,CPFD), aléatoire(MPQGA), duplication(DCP) avec deux types d'applications : la factorisation Cholesky [49] d'une matrice de grandeur variable jusqu'à 64 utilisée dans le domaine de l'analyse chimique quantique et le traitement de signal FFT. Un NSL ratio variable et des architectures variables qui sont définis comme suit :

M1 = indique une matrice <=30
M2 = indique une matrice entre 30 et 39
M3 = indique une matrice entre 39 et 64

NSL1 = indique une NSL < 2.5
NSL2 =indique une NSL entre 2.5 inclusivement et 3
NSL3 =indique une NSL entre 3 inclusivement et 3.5
NSL4 =indique une NSL entre 3.5 et 30

CPU8 qui indique une architecture jusqu'à 8 processeurs et ProcesseursNonBornées indique une architecture flexible de processeurs illimités. Nous créons cinq patrons pour la factorisation Cholesky suivants avec les informations dont nous disposons pour la factorisation de Cholesky.

Nom : MD(C_1)

Contexte :	Factorisation Cholesky TECH : {ProcesseursNonBornées} CapabilitySet = {NSL1}
Classement :	Configuration : NSL < 3.5 pour FSET
Problème :	FSET = {M1}
Solution :	L'algorithme d'ordonnancement ISET = {C} (C pour type d'ordonnanceur par cluster)

Nom : DCP(C_2)

Contexte :	Factorisation Cholesky TECH : {ProcesseursNonBornées} CapabilitySet = {NSL2, NSL3}
Classement :	Configuration : NSL entre 2.5 et 3.5 pour FSET
Problème :	FSET = {M2, M3}
Solution :	L'algorithme d'ordonnancement ISET {C} (C pour type d'ordonnanceur par cluster)

Nom : MCP(C_3)

Contexte :	Factorisation Cholesky TECH : {CPU8} CapabilitySet = {NSL1}
Classement :	Configuration : NSL entre < 2.5 pour FSET
Problème :	FSET = {M1}
Solution :	L'algorithme d'ordonnancement ISET {BNP} (BNP pour type d'ordonnanceur par processeurs bornées)

Nom : ETF(C_4)

Contexte :	Factorisation Cholesky TECH : {CPU8} CapabilitySet = {NSL1, NSL2, NSL3}
Classement :	Configuration : NSL entre < 3.5 pour FSET
Problème :	FSET = {M1, M2}
Solution :	L'algorithme d'ordonnancement ISET {BNP} (BNP pour type d'ordonnanceur par processeurs bornées)

Nom : BSA(C_5)

Contexte :	Factorisation Cholesky TECH : {CPU8} CapabilitySet = { NSL4}
Classement :	Configuration : NSL entre < 30 pour FSET

Problème : FSET = {M1, M2, M3}
Solution : L'algorithme d'ordonnancement ISET {APN}
(APN pour type d'ordonnanceur par processeur arbitrairement connectés)

L'utilisateur peut fouiller les patrons pour trouver les algorithmes compatibles avec les technologies souhaitées. Nous pouvons remarquer que selon le type d'algorithme utilisé (ie : BNP, APN, ProcesseursNonBornées) le classement est différent. Effectivement, la capacité d'avoir des gains de performances varie selon les algorithmes, mais avec les patrons, nous avons le type d'optimisation que nous pourrions avoir ainsi le type d'architecture utilisé. L'utilisation des patrons permet de conserver et de documenter le choix des architectures et des algorithmes avec les ententes de performance. La méthode des patrons permet aussi une manipulation plus facile des connaissances avec les ordonnanceurs.

À partir des patrons, nous pouvons créer un algorithme de type règle [108]. En effet, en regroupant les patrons C1 et C2 avec la signature indiquant la dimension de la matrice M1, M2 ou M3 pour choisir l'algorithme à utiliser. Nous utiliserons, par exemple, l'algorithme MD avec des matrices inférieures ou égales à 30. Avec des matrices plus grandes, nous utiliserons l'algorithme DCP. Noter que les patrons peuvent être avec n'importe qu'elle nombre de processeurs.

Nous pouvons utiliser l'approche pour le traitement de signal avec les FFTs.

SRL_1 = indique SRL entre 3 et 3.5
SRL_2 = indique SRL entre 4 et 4.5

Nom : MPQGA(F_1)
Contexte : FFT 16 points
TECH : {CPU_8}
CapabilitySet = {SRL_1}
Classement : Configuration : SRL1 pour FSET
Problème : FSET = {FFT_{16}}
Solution : L'algorithme d'ordonnancement ISET {Aléatoire}
(Aléatoire pour type d'ordonnanceur)

Nom : MPQGA(F_2)
Contexte : FFT 32 points
TECH : {CPU_8}
CapabilitySet = {SRL_2}
Classement : Configuration : SRL_2 pour FSET
Problème : FSET = {FFT_{32}}
Solution : L'algorithme d'ordonnancement ISET {Aléatoire}
(Aléatoire pour type d'ordonnanceur)

Nom : CPOP(F_3)
Contexte : FFT 16 points
TECH : {CPU_8}
CapabilitySet = {SRL_1}
Classement : Configuration : SRL_1 pour FSET
Problème : FSET = {FFT_{16}}
Solution : L'algorithme d'ordonnancement ISET {List}
(List pour type d'ordonnanceur)

Nom : CPOP (F_4)
Contexte : FFT 32 points
TECH : {CPU8}
CapabilitySet = {SRL_2}
Classement : Configuration : SRL_2 pour FSET
Problème : FSET = {FFT_{32}}
Solution : L'algorithme d'ordonnancement ISET {List}
(List pour type d'ordonnanceur)

Le dernier exemple montre l'utilisation d'algorithmes par duplication :

NSL_5 =indique une NSL < 1.5

NSL_6 =indique une NSL 1.5 et 2

NSL_7 = indique une NSL 2 et 2.5

N_{50} = indique application avec 50 noeuds.

Nom : CPFD(G_1)
Contexte : 50 noeuds
TECH : { ProcesseursNonBornées }
CapabilitySet = {NSL_5}
Classement : Configuration : NSL_5 pour FSET
Problème : FSET = {N_{50}}
Solution : L'algorithme d'ordonnancement ISET {C}
(C pour type d'ordonnanceur par cluster)

Nom : DCP(G_2)
Contexte : 50 noeuds
TECH : { Generique }
CapabilitySet = { NSL_6}
Classement : Configuration : NSL_6 pour FSET
Problème : FSET = {N_{50}}
Solution : L'algorithme d'ordonnancement ISET {TDB}
(TDB pour type d'ordonnanceur par duplication)

Nom : MCP(G_3)
Contexte : 50 noeuds
TECH : { CPU8 }
CapabilitySet = { NSL_7}
Classement : Configuration : NSL_7 pour FSET
Problème : FSET = {N_{50}}
Solution : L'algorithme d'ordonnancement ISET {BNP}
(BNP pour type d'ordonnanceur par processeurs bornées)

À partir de G_1 à G_3, nous pouvons remarquer qu'il y a différentes approches pour ordonnancer un graphe de 50 nœuds selon [112]. Nous remarquons que NSL_5 peut être obtenu avec des algorithmes de type cluster. Nous pouvons donc utiliser la grammaire pour stocker les patrons et extraire des informations pertinentes. L'importance est mise sur l'usage des patrons pour représenter les connaissances sur les algorithmes pour un non-expert. F_1 à F_4 indiquent des patrons capables d'ordonnancer des FFTs avec des métriques SRL allant de 3 à 4.5.

Nous avons aussi représenté les patrons sur connaissance apprise d'une plateforme [113] comportant plusieurs méthodes d'ordonnancement. Cette plateforme utilise les DAGs pour modéliser et ordonnancer l'algorithme de codage vidéo pour un ou plusieurs groupes d'images GOP. La signature, une caractéristique disponible qu'on appelle GPO peut être utilisée.

Nous utilisons les connaissances sur cinq méthodes d'ordonnancement d'un algorithme de codage vidéo à multiple vue. Celles-ci utilisent cinq approches différentes: la trame la plus prête (EF), la trame la plus référée (MRF), les trames les plus courtes en premier(SFF), le

plus court temps d'exécution attendu sur le chemin de référence(EE). Les trames les plus référées sur le chemin de référence (MR). FSET peut être GPO2, GPO3 selon la caractéristique de l'application à ordonnancer. Les résultats entendus est un facteur d'accélération (VAI) = {VA2, VA3, VA3, VA4, VA5, VA6, VA7, VA8, VA9} où I représente le facteur d'accélération. Par exemple, VA2 indique un facteur d'accélération de 2. Nous avons extrait 7 patrons (H1 à H7) à partir des informations disponibles :

Nom : Proc(H_1)

Contexte :	Codage vidéo à multiple vue TECH : {CPU4} CapabilitySet = {VA4}
Classement :	Configuration : VA <= 4 pour FSET
Problème :	FSET = {GPO2, GPO3}
Solution :	L'algorithme d'ordonnancement ISET = {EF}

Nom : Proc(H_2)

Contexte :	Codage vidéo à multiple vue TECH : {CPU8} CapabilitySet = {VA6}
Classement :	Configuration : VA <= 8 pour FSET
Problème :	FSET = {GPO2}
Solution :	L'algorithme d'ordonnancement ISET = {EF}

Nom : Proc(H_3)

Contexte :	Codage vidéo à multiple vue TECH : {CPU8} CapabilitySet = {VA7}
Classement :	Configuration : VA <= 8 pour FSET
Problème :	FSET = {GPO3}
Solution :	L'algorithme d'ordonnancement ISET = {EF}

Nom : Refs(H_4)

Contexte :	Codage vidéo à multiple vue TECH : {CPU10} CapabilitySet = {VA9}
Classement :	Configuration : VA <= 9 pour FSET
Problème :	FSET = {GPO3}
Solution :	L'algorithme d'ordonnancement ISET {MRF}

Nom : Ex(H_5)

Contexte :	Codage vidéo à multiple vue TECH : {CPU10} CapabilitySet = {VA9}
Classement :	Configuration : VA <= 9 pour FSET

Problème : FSET = {GPO2}
Solution : L'algorithme d'ordonnancement ISET {SSF}

Nom : SUM(H_6)
Contexte : Codage vidéo à multiple vue
TECH : {CPU10}
CapabilitySet = { VA9 }
Classement : Configuration : VA <= 9 pour FSET
Problème : FSET = {GPO2}
Solution : L'algorithme d'ordonnancement ISET {EE}

Nom : LEFT(H_7)
Contexte : Codage vidéo à multiple vue
TECH : {CPU10}
CapabilitySet = { VA9 }
Classement : Configuration : VA <= 9 pour FSET
Problème : FSET = {GPO2}
Solution : L'algorithme d'ordonnancement ISET {MR}

Nous remarquons que nous disposons plusieurs technologies CPU4, CPU8 et CPU10 avec certains résultats de gain d'accélération. Nous voyons que pour GPO3, nous pouvons avoir un gain de 9 avec 10 processeurs, un gain de 7 avec 8 processeurs et un gain de 4 avec 4 processeurs. Nous remarquons qu'avec 4 processeurs, nous obtenons un gain de 4 avec un GPO2 et GPO3.Nous pouvons avoir sous forme de patron, les algorithmes à utiliser selon le groupe d'images. Nous amenons une nouvelle façon de voir et de manipuler l'information par rapport à l'ordonnancement. Un utilisateur peut filtrer les patrons selon l'architecture choisi, le facteur d'accélération. Les patrons montrent aussi leurs caractéristiques propres Par exemple H_7 utilise

Nous pouvons donc conserver une trace des choix d'architectures et algorithmiques. Les patrons sont utiles pour faire une première sélection d'algorithmes capables de résoudre la problématique. Les patrons sont utilisés pour la phase du filtre 1 avec les méthodes mentionnées dans la section 5.2.3. De plus, nous pouvons faire des choix d'architectures avec les patrons.

5.4 Conclusion

Nous avons montré comment l'utilisation de langages et des grammaires des patrons peut capturer et manipuler des connaissances au sujet de la résolution des problèmes d'ordonnancement de tâches. D'après notre expérience, c'est la première tentative globale pour gérer l'ordonnancement des connaissances de l'algorithme avec des motifs. D'autres travaux ne couvrent pas cet aspect particulier ou proposent des études sur les modèles qui étaient trop génériques ou applicables à des domaines autres que les algorithmes de planification. Notre méthodologie peut être appliquée pour capturer des connaissances sur des algorithmes de planification et peut aussi être employée pour d'autres patrons externes non reliés à l'ordonnancement, tels que les modes de capture décisionnelle d'architectures décrits par Harrison et autres [92]. La grammaire choisie est polyvalente et peut représenter les algorithmes, les architectures et d'autres modèles. Cette utilisation des langages de patrons avec la grammaire et les règles d'association ouvre des possibilités infinies. Ce système est un complément parfait à l'analyse comparative, et il peut être facilement intégré dans les outils de conception.

Nous avons présenté cinq modèles qui peuvent être produits par nos méthodes, mais les possibilités sont infinies. Nous avons montrés différents patrons pour résoudre des types d'applications telles que FFT, factorisation et encodage vidéo avec différent type d'algorithmes.

Nous nous sommes concentrés sur la souplesse de l'environnement qui peut être intégré avec d'autres modèles ou environnements. Une grammaire flexible peut être utilisée à cette fin. Cependant, les langages de patrons sont généralement représentés dans un format textuel, parfois avec des graphiques. Nous avons par ailleurs introduit des ensembles d'informations dans le langage des patrons. Chaque ensemble représente une liste de règles d'association. Ces ensembles sont utiles dans la production de patrons. Nous utilisons une méthode de classement qui peut être associée à un outil de filtrage d'informations appliqué aux règles d'association. De plus, nous employons deux ensembles représentant les forces et les faiblesses d'un modèle. Cette approche offre une flexibilité en matière de principe de règles d'association. Nous pouvons utiliser une séquence de patrons qui peut représenter des

alternatives, les options et les architectures. Toutefois, l'utilisateur ou les outils doivent connaître les limites de notre méthodologie. Le classement dont nous nous servons représente un guide pour un patron; les classements doivent être basés sur des estimations et des spécifications déterminées par un expert.

Le prochain chapitre revient sur les concepts présentés jusqu'à maintenant et montre les possibilités et la synergie des modules proposés dans un nouvel engin d'extraction et de gestion des connaissances. La synergie des modules avec les intervenants y est démontrée.

CHAPITRE VI

MODELISATION ET EXTRACTION DE CONNAISSANCES A LA DOCUMENTATION POUR LES ALGORITHMES D'ORDONNANCEMENT DE TYPE LISTE

Ce chapitre revient sur les concepts relatifs à l'extraction et la représentation des méthodes d'ordonnancement en présentant un engin de connaissance plus élaboré que la proposition du chapitre II. Nous présentons donc ici un outil capable de manipuler et de présenter les connaissances avec la combinaison de langage de patrons et des règles d'associations.

Nous proposons un moteur d'extraction et de gestion des connaissances pour ce domaine d'activité et une autre considération importante, celle d'enregistrer et de tracer les connaissances utilisées lors de la sélection et l'application des algorithmes. Notre stratégie consiste à montrer qu'en plus de réaliser la gestion de connaissance, la plateforme peut être intégrée aux outils documentaires. Nous montrons un métamodèle pour modéliser et documenter les décisions avec différents points de vue qui incluent les patrons, les spécifications et les solutions. Éventuellement, cet engin peut être utilisé dans des plateformes automatisées de conception et procurer des ensembles de solutions.

Nous dévoilons différents ensembles de données qui peuvent être manipulés : les données brutes, les règles d'association, les patrons et les diagrammes. Nous illustrons comment un processus à deux phases, deux filtres de connaissance, peut chercher les informations pertinentes à un problème tout en donnant différentes vues sur les possibilités offertes. Le premier filtre utilise trois méthodes pour extraire les candidats pour le deuxième filtre : règle de concept, assortiment et séquences de patrons. Les séquences de patrons peuvent représenter les alternatives, les options et les architectures requises. Le deuxième filtre utilise les informations recueillies par le premier filtre pour effectuer un banc d'essai. Nous proposons d'ajouter la cartographie par règles avec le deuxième filtre. De plus, nous montrons comment enregistrer les décisions prises par les filtres. Ceci permet une intégration

de connaissances dans un système documentaire. En fait, nous proposons d'utiliser des diagrammes de Hasse qui peuvent guider les usagers de notre plateforme. Les experts en ordonnancement disposent aussi de notre plateforme et bénéficient de nouvelles méthodes pour comparer les algorithmes sur la base de règles d'association, de patrons ou des deux. L'interface de notre outil fournit une façon d'enregistrer et d'extraire les connaissances pour d'autres outils qui peuvent être intégrés. Nous focalisons sur la flexibilité de notre engin qui peut être intégré avec les autres patrons et outils.

Dans ce chapitre, nous commençons donc par examiner les approches que nous avons utilisées dans le travail passé; ensuite, nous fournissons et élaborons quelques exemples. Par conséquent, nous cherchons à tirer parti de notre travail avant de présenter un moteur de connaissances complet conçu pour les algorithmes d'ordonnancement. Cet outil peut être considéré comme un complément de bancs d'essai et les outils de documentation actuels pour aider le concepteur et l'expert dans la zone de planification de tâches.

Dans cette optique, de quelle manière devrions-nous représenter les connaissances hétérogènes sur les ordonnanceurs pour aider notre concepteur d'applications à faire son choix selon son application et l'architecture ciblée ? Dans un second temps, est-ce que nous pouvons mettre en place une méthodologie, une approche pour choisir automatiquement un ordonnanceur statique pour notre concepteur tout en fournissant de façon efficace les raisons de ce choix de conception logiciel ?

6.1 Environnement et exemples proposés

Nous commençons avec un aperçu de notre environnement et de ses capacités les plus saillantes. Nous revenons sur notre motivation et proposons un engin avec trois types d'usagers.

6.1.1 Motivation

L'outil proposé reflète les connaissances sur la planification des tâches et la façon de la gérer. Comme indiqué précédemment, il existe un grand nombre de connaissances sur les techniques d'ordonnancement des tâches. En outre, il existe une courbe d'apprentissage

abrupte pour un nouvel utilisateur dans ce domaine qui cherche à obtenir des informations par des moyens simples (c'est-à-dire sans avoir à gérer une quantité excessive d'informations). Chaque algorithme peut avoir ses propres forces et faiblesses et peut cibler une architecture spécifique. Les concepteurs disposent de nombreuses possibilités. Toutefois, nous voulons donner de nouvelles méthodes aux experts pour repérer facilement les informations relativement aux algorithmes d'ordonnancement. Nous voulons un moteur qui soit souple et évolutif concernant la connaissance des techniques de planification.

6.1.2 Introduction applicative

Avant de présenter notre moteur, nous introduisons quelques concepts employés dans notre environnement. Supposons que la nécessité du concepteur est de réaliser des applications variées et particulières sur un groupe de trois processeurs entièrement reliés. Nous définissons ces applications App_1 et App_2, notre concepteur D1 et un expert en planification SE1. Ce concepteur travaille au sein d'une équipe avec l'actionnaire SH1 qui est propriétaire de l'entreprise. Une décision doit être prise quant à la mise en œuvre de ces applications dans un délai donné tout en enregistrant les informations sur le détail des implémentations choisies pour référence ultérieure. Lors de la planification des tâches, le concepteur doit savoir quelles techniques utiliser; ce n'est pas une tâche facile. Actuellement, les outils ne sont pas flexibles, ils doivent traiter un grand nombre de connaissances ou en sont à un stade de recherche préliminaire [86]. L'objectif de ce chapitre est de montrer de quelle façon nous pouvons aider la conception, mais aussi comment le moteur proposé améliore la connaissance des techniques de planification et est évolutif.

6.1.3 Moteur

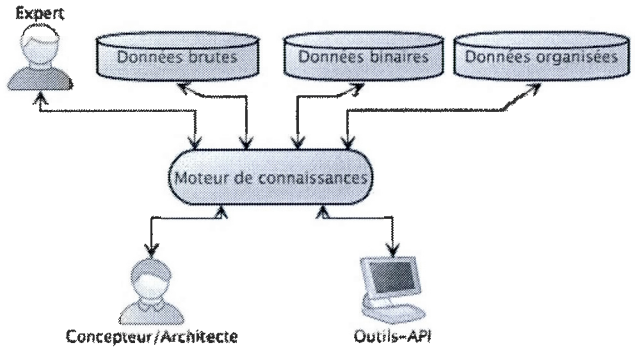
La figure 6.1 de la page suivante présente le schéma du moteur et de l'environnement de connaissances. Trois types d'utilisateurs sont pris en charge : expert, concepteur et outils. Six éléments clefs doivent être considérés :

1. Données brutes : comprend les données d'essai au banc, conceptions passées, tendances, documents et autres connaissances relatives aux connaissances d'experts comme SE1.

2. Designer : fournit l'application pour planifier avec ses contraintes. Cela peut être D1, notre concepteur décrit précédemment.

3. Expert : réalise les algorithmes d'ordonnancement ou il a une connaissance approfondie à leur sujet. Cela peut être SE_1.

Figure 6.1 : Survol du moteur et son interaction



4. Outils : environnement de conception, outil de conception et de documentation dans lequel la plateforme peut être incorporée. Notre approche peut être utilisée en conjonction avec d'autres outils.

5. Données binaires : données brutes qui sont utilisées dans le processus de découverte. Un format interne que nous avons proposé et qui peut être employé par nos outils.

6. Données de connaissances organisées : informations algorithmiques dans des formats tels que des règles, des modèles et des cartes de règles.

Le but du moteur est d'interagir avec l'environnement pour extraire des informations au sujet des algorithmes d'ordonnancement. De plus, il donne un aperçu des méthodes de planification qui peuvent être utilisées de manière efficace par l'utilisateur non-expert D_1 et SH_1. La plateforme peut être utilisée pour implémenter des applications, créer des algorithmes et documente ses décisions sur les algorithmes d'ordonnancement.

Les règles d'association peuvent être fixées à l'avance par un expert comme SE1 ou être découvertes. La section suivante traite de la méthodologie de découverte vue au chapitre IV.

6.1.4 Découverte des connaissances

Supposons que SE_1 dispose d'une centaine d'algorithmes d'ordonnancement. SE_1 sait que les algorithmes et les critères conduisent à des résultats génériques, mais quel algorithme répond à l'exigence de App_1 et App_2 de notre designer D_1? Comment manipuler et extraire des connaissances à propos de ces algorithmes de façon structurée? L'outil peut être réglé en mode de découverte pour en savoir plus sur les algorithmes d'ordonnancement.

Les signatures peuvent être groupées pour créer des ensembles d'applications avec des métriques de performances communes. Avec la méthode décrite, SE_1 expert peut envisager un groupe d'applications et les analyser du point de vue des règles d'association pour tester et améliorer les algorithmes d'ordonnancement pour le groupe. Toutefois, la règle d'association est considérée comme une connaissance gérée. Dans la section suivante, nous nous concentrons sur l'interface utilisateur et sur ce que D_1 peut faire avec notre environnement.

6.1.5 Point de vue designer/architecte

La méthodologie proposée considère à la fois le concepteur et les contraintes technologiques imposées, l'impact des métriques et la hiérarchie des tâches. Elle fonctionne comme suit : après la création d'une représentation graphique de sa conception, le concepteur spécifie les entrées et sorties, l'architecture de mise en œuvre et les caractéristiques souhaitées du système telles la tolérance de panne, le nombre de processeurs et la vitesse. Cette information sert ensuite à trouver un algorithme d'ordonnancement approprié pour un GAD donné, comme décrit ci-dessous.

L'utilisateur est une personne qui veut mettre en œuvre une application et qui utilise un environnement de développement intégré (IDE) pour le faire. Ainsi, l'interface utilisateur permet au non-expert d'exploiter efficacement la connaissance d'ordonnancement organisée.

Le concepteur, D_{ij}, entre une architecture et un GAD à utiliser pour la conception ainsi que la liste des paramètres tels que les exigences d'application comme la tolérance de panne, les processeurs et la vitesse limite. Ensuite, un algorithme qui répond aux besoins exprimés lui est retourné à partir du contenu de la base de connaissances. Pour atteindre cet objectif, la première étape extrait les caractéristiques de la GAD et identifie les algorithmes d'ordonnancement compatibles dans la base de connaissances. Puis, deux étages de filtrage sélectionnent les candidats retenus, ils utilisent l'analyse de décomposition, la résolution des processus [12] et le concept d'exploration [97]. Les deux premières techniques permettent de trouver des candidats qui répondent aux exigences et d'élaborer un tableau de bord pour obtenir la meilleure solution; la troisième consiste à générer des données avec des candidats potentiels en vue de résoudre un problème donné. Puis, un candidat est sélectionné.

Filtre 1

Filtre 1 transforme d'abord les caractéristiques du GAD, l'architecture et les contraintes dans une liste d'attributs. Trois méthodes existent pour extraire des connaissances des algorithmes supportés : concepts de règles (chapitre II), ensemble assorti (chapitre V) et séquences (chapitre V).

a) Concepts de règles

Pour identifier les règles, nous utilisons l'algorithme Bordat [99] qui établit les relations entre des éléments de relations binaires. Ensuite, les graphes conceptuels correspondants (diagrammes de Hasse) sont produits pour structurer les concepts généralisateurs et préciser les relations entre les algorithmes. Enfin, un diagramme de Hasse est conçu pour extraire un groupe concept GA' qui possède des attributs communs de l'ensemble GA de tous les algorithmes disponibles.

Cette phase de filtrage comprend également des anti-patrons pour exclure des algorithmes connus pour ne pas s'appliquer au problème proposé. Ceci est accompli grâce à la connaissance des faiblesses des algorithmes et la vérification des antimodèles associés. Après cet élagage, GA' deviendra GA" : une liste de candidats. La figure 6.2 illustre plusieurs concepts extraits de la matrice de 18 techniques et des GAD signature. Notez que la dernière

ligne de la figure 6.2 n'a pas d'algorithme. Ceci indique qu'il n'y a pas de technique qui rencontre le classement pour les signatures GAD qui sont considérées.

Par exemple, la figure 6.2 montre que, compte tenu de notre première application App1 qui possède une signature GAD N_4 (ce qui signifie une demande de 80 à 99 nœuds), n'importe lequel des algorithmes CPOP, HEFT, HEFTA1B0, HEFTA2B0, HEFTA3B0 et PETS peut être utilisé pour planifier l'application. La seconde application App_2, qui possède une signature GAD C_0, donne 12 candidats.

Figure 6.2 : Vue d'un diagramme de Hasse avec 80 % de niveau de confiance

b) Ensemble assorti

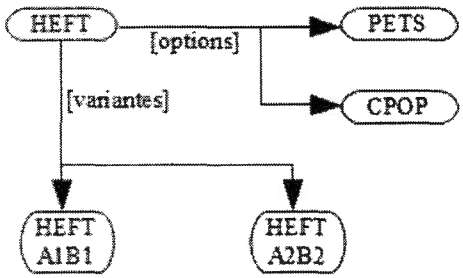
La seconde approche, l'ensemble assorti, étend les possibilités et la flexibilité de la première méthode. Plus précisément, Filtre 1 peut définir et exploiter un classement des modèles de planification disponibles comme présenté au chapitre VI. Ce classement spécifie la façon dont nous faisons confiance à un algorithme d'ordonnancement de fournir un résultat de

performance donné. Nous recueillons des informations sur les performances et les règles pour établir le classement. Nous pouvons définir différents seuils et conditions de classement. Par exemple, un facteur d'accélération de 20 % avec un niveau de confiance à 80 % de la règle peut être choisi. Le problème d'ordonnancement devient alors le traitement d'un ensemble d'attributs de règles d'association où la prémisse de chaque règle est prise à partir de la signature GAD et un niveau de confiance de 80 %. Nous pouvons définir un ensemble FSET qui répertorie les modèles de signature pour lesquels l'algorithme s'applique, par PETS qui réponds au classement pour les signatures suivantes : {N_3, N_4, C_0, N_2C_1, N_3C_1, N_4C_1, N_3C_2, N_4C_2}. Ainsi, en ce qui concerne App1 = { N_4C_2 } et App2 = { N_0C_0}, PETS peut être choisis en raison de N_4 et C_0 respectivement. Nous pouvons également définir un ensemble WSET pour la liste des motifs de signature GAD dont l'algorithme fonctionne mal, FSET et WSET peuvent être utilisés pour filtrer les algorithmes. Nous appliquons la procédure d'ensemble assorti proposée au chapitre V dans Filtre 1. En bref, la méthode cherche les signatures GAD présentes dans FSET et non dans WSET. En prenant l'application de la factorisation Cholesky pour une matrice de grandeur 35, le filtre 1 donne deux patrons (C2 et C4) sur les cinq disponibles avec différentes performances avec un classement NSL<3.5.

c) Séquences

La dernière méthode pour extraire les algorithmes candidats consiste à utiliser l'approche de séquence de patrons du chapitre V. Pour obtenir une séquence, nous avons besoin d'un algorithme de référence. Nous pouvons rechercher la première compatibilité avec la signature de l'application et chercher une séquence de motif. Le programme va exécuter la grammaire associée pour produire une règle en vue d'obtenir des informations sur les options, les alternatives et les architectures compatibles. Par exemple, en utilisant HEFT comme premier algorithme, nous obtenons la séquence 'HEFT: HEFTA1B1' 'HEFT: HEFTA2B2', 'CPOP', 'PETS' comme illustré à la figure 6.4, basé sur [91] . Les deux premiers éléments constituent deux variantes de HEFT et les troisième et quatrième éléments sont des options comme illustré avec les patrons du dernier chapitre. La figure 6.3 montre les candidats ci-dessus sous forme de diagramme de séquence.

Figure 6.3 : Diagramme de séquence



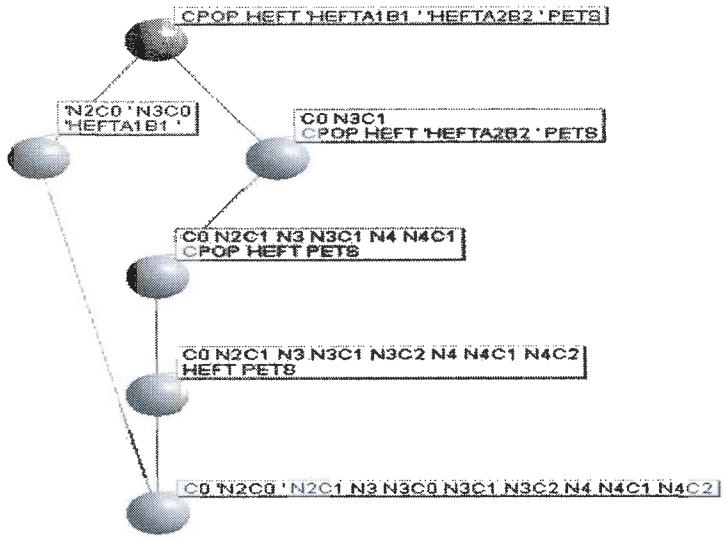
Chaque élément dans le diagramme peut être un modèle en dehors d'une technique d'ordonnancement. Notez que le diagramme fournit des informations sur les variantes et les options de planification pour le concepteur et peut facilement enregistrer la décision prise à propos de la sélection d'une approche algorithmique.

Les trois méthodes peuvent sauvegarder les décisions prises. En outre, nous pouvons extraire un diagramme de Hasse des deux dernières méthodes. Extraire les concepts à partir d'une matrice composée de chaque patron comme objet à l'aide de chaque FSET. La figure 6.4 montre l'exemple d'un diagramme de Hasse obtenu à partir d'un ensemble de motifs de FSET présenté au chapitre précédent.

Il est à noter que Filtre 1 ne garantit pas un candidat. En effet, il recherche des candidats au sein de contraintes telles que le niveau de confiance, le classement et la base de la connaissance.

Lorsque Filtre I est terminé, tous les candidats sont propagés à la phase de Filtre 2 où ils sont comparés et éventuellement comparés. Ensuite, nous sélectionnons l'algorithme final retenu. Le tableau 6.1 montre les candidats pour nos deux applications.

Figure 6.4 : Diagramme de Hasse sur les FSET



Nous remarquons que HEFT, PETS et CPOP sont candidats pour les deux applications parce que leurs patrons matchent les signatures d'application (voir la discussion sur les ensembles assortis avec PETS). D'un autre côté, HEFTA2B2 = $\{C_0, N_3C_1\}$ n'est pas un bon candidat pour App_1 parce qu'il n'y a pas de compatibilité entre les patrons et la signature App_1= {N4C2}.

Tableau 6.1 Choix des candidatures de Filtre 1

Application	**Choix du filtre avec cinq patrons algorithmes**
App1	HEFT, PETS, CPOP
App2	HEFT, PETS, CPOP, HEFTA2B2

Filtre 2 et solution proposée

Filtre 2 produit des indicateurs tels que le *gain*, NSL, SRL, ADP et NB pour chaque candidat. Différentes estimations sont réalisées. Elles peuvent comprendre les valeurs pire, médiane et moyenne pour les coûts de calcul. Le concepteur peut alors utiliser ces paramètres pour choisir un algorithme vainqueur pour son design. Les informations de Filtre 2 peuvent également être utilisées pour la documentation sur un GAD et toute application particulière.

Le tableau 6.2 montre la métrique d'accélération pour les applications App1 et App2 pour nos candidats patrons choisis au filtre 1. (NS) indique que ce modèle n'a pas été candidat. L'astérisque * indique le vainqueur sur la base de la métrique de vitesse.

Tableau 6.2 Filtre 2 avec métrique de vitesse

Application	**P1 HEFT**	**P2 CPOP**	**P3 PETS**	**P5 HEFT A2B2**
App1	157	106	200*	85 (NS)
App2	122	84	128*	26

Le principe est applicable pour l'application factorisation Cholesky. Nous pouvons choisir les algorithmes DCP(C2) et ETF(C4) selon les besoins de l'usager. Le DCP possède une moyenne NSL plus basse que l'algorithme ETF pour une matrice de 35.

Nous verrons à la section suivante la capture de la décision finale.

6.1.6 Capture de la décision finale

Comme les deux dernières approches – ensemble assorti et séquence – permettent de fournir des algorithmes candidats avec une représentation de patron, nous pouvons tirer parti de la notion de langage de patrons pour fournir au concepteur D_i beaucoup d'informations supplémentaires et utiles d'une manière efficace. En effet, le modèle Vee peut également aider à gérer la complexité de la phase de développement. En particulier, il peut gérer les questions quoi? pourquoi? qui? relatives à un système et ses sous-éléments (par exemple, pourquoi utilisons-nous cet algorithme ou ce type d'architecture?). De plus, le modèle peut rendre compte des parties prenantes SH_i, de la gestion des risques et de la résolution de problème. Notre moteur de connaissances peut facilement fournir des informations sur les techniques d'ordonnancement utilisées et leurs options. Nous pouvons inclure le diagramme de séquence et le diagramme de Hasse dans le processus de documentation. Cela peut fournir à l'architecte une meilleure compréhension des techniques d'ordonnancement et du pourquoi, et le moteur de connaissance peut fournir des traces de la décision. Dans le modèle Vee+, les étapes d'entrée et sortie sont deux étapes importantes pour l'analyse de décomposition et la

résolution des processus. La première étape nécessite des spécifications comme critères d'entrée et certaines sorties sont nécessaires en tant que documentation. Les critères d'entrée peuvent être passés au filtre comme des éléments des prémisses de règles ou conséquents. Lago et Avgeriou [85] ont défini deux types de connaissances :

- Application générique : l'expertise qui s'applique à un ou plusieurs domaines et qui peut venir de leurs expériences antérieures.

- Application spécifique comme App1 et App2 : la connaissance des solutions et procédés utilisés pour mettre en œuvre un système particulier.

Notons que l'interface utilisateur peut fournir des informations complémentaires sur les décisions d'application spécifiques. Notre approche tente de faire correspondre un ensemble de connaissances spécifiques de l'application et de créer des connaissances génériques. En outre, nous fournissons l'interface expert pour améliorer la base de connaissances.

6.2 Interface Expert

L'expert est préoccupé par l'optimisation et l'ordonnancement des applications. Cette personne utilise l'interface expert pour modifier et mettre à jour la base de données d'algorithmes disponibles pour l'utilisateur ou un outil. L'interface fournit également un mécanisme pour améliorer et valider des méthodes de planification. Une interface expert est utilisée pour acquérir et organiser les connaissances sur les algorithmes et les technologies. Elle est employée pour tester et valider les algorithmes avant leur inclusion dans une base de connaissances structurée qui est disponible pour le concepteur et les outils. Il existe de nombreuses facettes à l'ajout de connaissances à cette interface. Les algorithmes d'ordonnancement disponibles sont créés par des experts qui connaissent les plateformes utilisées avec leurs particularités (par exemple, la fréquence de fonctionnement, la tension d'alimentation, etc.). Les algorithmes sont normalement conçus pour des types spécifiques d'architecture et de paramètres à optimiser, et une interface expert et une interface utilisateur de première ligne sont utilisées pour gérer et exploiter ces connaissances.

L'expert peut utiliser une carte de règles pour comparer des algorithmes. Nous avons présenté

au chapitre V le concept de carte de règles et son application aux algorithmes d'ordonnancement consultés individuellement ou en groupes. Cette représentation bidimensionnelle des forces et des faiblesses d'un algorithme d'ordonnancement par une carte de couleur permet l'identification facile de problème quand nous l'utilisons. En plus, la même technique présentée dans la phase Filtre 1 peut être utilisée par l'interface expert pour afficher des informations sur le diagramme de Hasse. En effet, nous pouvons nous concentrer sur l'ensemble des éléments GA' et produire une carte de règles pour ce groupe d'algorithmes. Ces outils fournissent une flexibilité pour afficher et accroître la compréhension de l'expert. Nous verrons maintenant comment créer de nouveaux algorithmes basés sur notre moteur.

6.3 Classe d'algorithme par règles

Nous avons proposé une approche de classe de règles pour créer de nouveaux algorithmes [108]. SE_i peut utiliser cette approche pour créer de nouveaux algorithmes pour ses besoins. Nous pouvons étendre l'approche avec des patrons de signature. Comme indiqué précédemment au chapitre VI, nous avons FSET et WSET qui contiennent des modèles de signature lorsque l'algorithme s'applique ou non. Supposons que nous voulons créer un nouvel algorithme de classe de règle. Nous adoptons une approche de planification et nous considérons son FSET pour déterminer quand il faut utiliser l'algorithme de référence d'une application donnée. Ensuite, nous pouvons ajouter de l'information complémentaire qui tient compte des autres FSET d'autres algorithmes qui ne sont pas dans le FSET ou WSET de la technique de référence. Cette méthode tente de créer un algorithme générique. Nous prenons cette information et les règles du groupe avant d'appliquer les méthodes illustrées dans [108], Par exemple, si nous avons deux algorithmes avec leurs FSET correspondantes : Algo_A avec FSET = $\{N_0\}$ et Algo B avec FSET = $\{N_1\}$. Ensuite, nous pouvons créer un nouvel algorithme Algo_C qui exécutera ALGO_A quand l'entrée DAG aura une signature de N_0 et exécutera Algo_B lorsque la signature DAG sera N_1. Ainsi, nous créons une exécution de l'algorithme conditionnelle basée sur les antécédents correspondants. Cela crée un algorithme composite à base de modèles.

6.4 Interface Outils

Les techniques d'ordonnancement sont principalement utilisées pour la planification des tâches, mais une autre question apparaît à propos de l'enregistrement et la traçabilité. Comme le moteur proposé extrait et gère le savoir, il peut être utilisé à l'intérieur d'un outil d'organisation. Capilla et autres [84] présentent un métamodèle pour modéliser et documenter les décisions d'architecture avec différentes perspectives qui comprennent les modèles, les exigences et les solutions d'architecture. Par la suite, le moteur peut être utilisé dans un environnement de conception automatique pour fournir des solutions.

6.5 Conclusion

Dans ce chapitre, nous avons présenté un moteur de connaissances pour les algorithmes d'ordonnancement. Selon notre expérience, notre approche diffère des autres travaux, car elle se concentre sur la gestion des connaissances pour la sélection de l'algorithme d'ordonnancement et la création. Les auteurs les plus proches sont les suivants : Santos et Koskimies [94] ont montré qu'il existe un manque de méthodes et d'outils pour enregistrer et exploiter les décisions de conception architecturale; Ils ont proposé aussi une couche supplémentaire de langage-patron pour des modules réutilisables; Lago et Avgeriou [85] traitent des méthodes et des outils qui peuvent extraire, découvrir et partager les connaissances, mais pour un usage général ; Borchers [95] introduit une approche pour capturer les connaissances des développeurs et des experts du domaine de l'application dans le développement de logiciels, l'interaction humain-ordinateur et les domaines d'application; Keutzer et Mattson [86] montrent comment les modèles de conception peuvent être utilisés dans les éléments logiciels réutilisables. Afin de comparer des algorithmes, la majorité des auteurs d'ordonnancement utilisent des graphiques et des indicateurs pour montrer les performances [19], [20], [21], [22], [26], [18], mais ils ne proposent pas de techniques de documentation et de cadre de gestion des connaissances.

Nous avons montré différents types de données qui pourraient être manipulés : crues, binaires, règles, modèles et diagrammes. Nous avons utilisé les données brutes pour construire des règles d'association qui pourraient être utilisées pour identifier les tendances et

produire des diagrammes. Nous avons montré comment les processus de découverte fonctionnent en présentant un processus en deux étapes avec deux filtres de connaissances, où le premier fournit des solutions candidates à la seconde. Les candidats peuvent être extraits avec trois méthodes : les règles de concepts, les ensembles correspondant et les séquences. Nous pouvons utiliser la séquence de motifs qui peut représenter l'alternative, l'option et l'architecture utilisées. D'autre part, nous avons montré comment enregistrer le processus de prise de décision par les filtres. Ainsi, il est facile d'intégrer les conclusions de connaissances dans un système de documentation. En fait, des diagrammes et des schémas de Hasse peuvent guider et diriger l'utilisateur.

L'expert dispose des mêmes outils que l'utilisateur et profite de nouvelles méthodes pour comparer les algorithmes basés sur des règles d'association, les modèles ou les deux. L'interface de l'outil fournit un moyen facile pour enregistrer et extraire des connaissances de données pour d'autres outils qui peuvent être intégrés dans notre moteur. Nous nous sommes concentrés sur la souplesse d'un environnement qui peut être intégré avec d'autres modèles et outils.

En conclusion, nous avons expliqué comment utiliser des règles d'association et l'extraction de données pour trouver et représenter de nouvelles connaissances. Notre méthodologie peut être utilisée dans d'autres travaux qui incluent un aspect global de conception et d'autres patrons qu'algorithmiques. Les trois interfaces de notre moteur peuvent être utilisées pour exploiter et accroître les connaissances concernant les algorithmes d'ordonnancement. Notre moteur permet la gestion et la création de connaissances. C'est le complément idéal de référence et d'analyse des outils, outils de conception et IDE.

Le dernier chapitre récapitule l'ensemble du travail de recherche présenté et conclut ce document en donnant les nouvelles possibilités offertes pour la manipulation, la gestion et la représentation des algorithmes d'ordonnancement.

CHAPITRE

DISCUSSION ET CONCLUSION

L'ordonnancement et l'allocation des tâches sont des problèmes bien connus. Malheureusement, les solutions à ces problèmes elles sont d'une complexité exponentielle. Les algorithmes disponibles utilisent les valeurs estimées comme médianes, pires des cas, et meilleures valeurs de cas pour l'évaluation des coûts d'exécution et de communication. Ces valeurs estimées ne sont pas parfaites et auront une incidence sur les résultats algorithmiques obtenus. De plus, les architectures et le type d'application peuvent avoir un effet. Nous avons proposé d'utiliser l'analyse formelle de concept comme un outil d'analyse de haut niveau afin de réduire la complexité du problème d'ordonnancement et l'impact des valeurs estimées. Notre approche diffère des autres auteurs parce qu'au lieu de l'approche statistiques, nous explorons d'autres méthodes d'analyse et d'extraction de connaissances à partir des données brutes. Nous avons introduit un cadre qui est flexible et qui exploite des connaissances organisées quant aux algorithmes d'ordonnancement.

Nous avons également proposé d'utiliser l'exploration de données pour extraire des connaissances des données d'application et évaluer l'impact d'un algorithme d'ordonnancement sur elles. Nous avons montré que les règles extraites peuvent être utiles dans de nombreuses façons d'évaluer l'ensemble des données et les performances d'un algorithme d'ordonnancement. Nous avons ensuite introduit le concept d'une règle Δ pour comparer les algorithmes; notre objectif était de repérer les forces et les faiblesses de chaque algorithme et de fournir ces informations à l'expert. À cet égard, la méthodologie mise en place est souple et exploite efficacement les règles visant à accroître les connaissances au sujet des algorithmes d'ordonnancement.

Nous avons montré comment utiliser les connaissances à propos des forces et faiblesses des algorithmes pour créer une nouvelle classe d'algorithmes d'ordonnancement basée sur les règles d'association. Cette classe pouvait être utile à bien des égards pour améliorer la performance d'un algorithme d'ordonnancement. Nous avons utilisé la règle Δ pour comparer les algorithmes et extraire des informations concernant les anti-patrons. À cet effet, cette nouvelle méthode d'ordonnancement est souple et efficace et elle permettait de produire des algorithmes pour réduire les ressources *makespan* et les processeurs [108].

Par ailleurs, nous avons proposé une technique, la carte de règles, pour la comparaison d'algorithmes d'ordonnancement et nous avons fourni des exemples d'utilisation. Elle comporte l'identification facile des signatures des applications sur l'axe x et les algorithmes ou des métriques sur l'axe y sous forme de matrice avec différents niveaux de couleur selon le niveau de confiance. En conséquence, l'interprétation de la carte de règles dépend de la nature des données que nous utilisons.

L'utilisation de visualisation de la carte de règles augmente la connaissance du concepteur quant aux différents algorithmes d'ordonnancement et les différentes combinaisons de variables x et y sont presque infinies. L'utilisation de règles d'association pour créer des listes de règles et la méthode de visualisation proposées avec la cartographie des règles fournit une nouvelle perspective pour comprendre les algorithmes d'ordonnancement; elles peuvent être un complément bienvenu aux repères traditionnels.

Nous avons montré ensuite comment l'utilisation des langages et des grammaires de patrons permettent de capturer et de manipuler des connaissances relatives à la résolution de problèmes d'ordonnancement des tâches. D'après notre expérience, il s'agit de la première tentative globale pour gérer les connaissances des algorithmes d'ordonnancement avec des patrons. D'autres travaux ne couvraient pas cet aspect particulier ou proposaient des modèles qui sont trop génériques ou seulement applicables à d'autres domaines. La grammaire qui a été choisie est polyvalente et peut représenter l'algorithme, les modèles architecturaux et autres. Cette utilisation des langages de patrons avec les règles de grammaire et d'association ouvre des possibilités infinies. Nous avons présenté cinq patrons qui peuvent être produits par nos méthodes. Nous avons adapté certains concepts de langage de patrons et de grammaire.

Cependant, les patrons sont généralement dans un format textuel, parfois avec des graphiques; nous avons introduit quelques ensembles. Chaque ensemble représente une liste de règles d'association. Ces ensembles sont utiles pour la réalisation de patrons. Nous avons montré une méthode de classement qui peut être utilisée avec un outil pour filtrer les informations à partir des règles d'association. De plus, les deux ensembles proposés représentent les forces et faiblesses d'un patron. Cette approche offre une souplesse à l'égard des règles d'association. Nous pouvons employer une séquence de patrons qui peuvent représenter les alternatives, les options et les architectures requises. Toutefois, l'utilisateur ou les outils doivent connaître les limites de notre méthodologie. Le classement utilisé est un guide et est basé sur des estimations et devis d'un expert.

Nous avons présenté un moteur de connaissances pour la planification de tâches. Selon notre expérience, notre approche diffère des autres travaux et se concentre sur les méthodes de planification. Nous avons montré différents types de données qui peuvent être manipulés : premières, binaires, règles, modèles et diagrammes. Nous avons utilisé les données brutes pour construire des règles d'association qui peuvent être utilisées pour identifier les patrons et produire des diagrammes. Nous avons illustré comment un processus de découverte peut être réalisé en deux étapes de deux filtres de connaissances, où le premier filtre donne les candidats au second filtre. Les candidats sont extraits par trois méthodes : les concepts de règles, l'assortiment d'ensembles et les séquences. Nous avons utilisé les séquences de patron pour donner les alternatives, options et architectures requises. Nous avons démontré que nous pouvions ajouter une carte de règles pour augmenter les connaissances. De plus, nous avons montré comment enregistrer le processus décisionnel par les filtres, ce qui facilite l'intégration des résultats des connaissances dans un système de documentation. En fait, les diagrammes de Hasse et les modèles peuvent guider et orienter l'utilisateur. L'expert dispose des mêmes outils que l'utilisateur et détient de nouvelles méthodes pour comparer des algorithmes basés sur des règles d'association, des modèles ou les deux. L'interface de l'outil proposé fournit un moyen simple pour extraire des données pour la connaissance d'autres outils. Nous nous sommes concentrés sur la flexibilité d'une plateforme qui peut être intégrée avec d'autres modèles et outils.

En conclusion, nous avons expliqué comment utiliser des règles d'association et d'exploration de données à extraire et représenter de nouvelles connaissances. Notre méthodologie peut être appliquée pour saisir les connaissances sur les algorithmes d'ordonnancement et être utilisée dans d'autres travaux couvrant un contexte plus global tel que les patrons pour capturer les décisions architecturales. Les trois interfaces de notre moteur de connaissance peuvent être utilisées pour exploiter et accroître les connaissances au sujet des algorithmes d'ordonnancement.

BIBLIOGRAPHIE

[1] A. Agarwal, «The Tile processor: A 64-core multicore for embedded processing», *Proceedings of High Performance Embedded Computing Workshop*, pp.1-18, 2007.

[2] G. Kurian, J. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. Kimerling et A. Agarwal, «ATAC: a 1000-core cache-coherent processor with on-chip optical network», *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pp. 477-488, 2010.

[3] D. Wentzlaff et A. Agarwal, «Factored operating systems (fos): the case for a scalable operating system for multicores», *ACM SIGOPS Operating Systems Review,* vol. 43, n° 12, pp. 76-85, 2009.

[4] M. Laurence, «Introduction to Octasic Asynchronous Processor Technology», , *2012 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pp. 113-117, 2012.

[5] E. Wynters, «Parallel processing on NVIDIA graphics processing units using CUDA», *Journal of Computing Sciences in Colleges,* vol. 26, n° 13, pp. 58-66, 2011.

[6] D. Smith, «VHDL and Verilog compared and contrasted-plus modeled example written in VHDL, Verilog and C», *Proceedings of Design Automation Conference 1996, 33rd*, pp. 771-776, 1996.

[7] J. Lapalme, E. Aboulhamid, G. Nicolescu, L. Charest, F. Boyer, J. David et G. Bois, «NET framework-a solution for the next generation tools for system-level modeling and simulation», *Proceedings of Design, Automation and Test in Europe Conference and Exhibition,* pp. 732-733, 2004.

[8] L. Kriaa, «Execution Models», *Book on Global Specification and Validation of Embedded Systems: Integrating Heterogeneous Components* , 2007.

[9] T. Schattkowsky, J. Hausmann et G. Engels, «Using UML activities for system-on-chip design and synthesis», *Model Driven Engineering Languages and Systems,* pp. 737-752, 2006.

[10] B. Boigelot, B. Mathieu, E. Quentin et E. Stéphane-DEA, « Systèmes programmés enfouis », *Notes de Cours,* 2005.

[11] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz et T. Euler, «Yale: Rapid prototyping for complex data mining tasks», *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 935-940, 2006.

[12] H. Mooz et K. Forsberg, A Visual Explanation of Development Methods and Strategies Including the Waterfall, Spiral, Vee, Vee+ , Vee++ Models, Melbourne, Australia: *Proceedings of the International Council for Systems Engineering (INCOSE) Conference*, 2001.

[13] H. Topcuoglu, S. Hariri et M.-Y. Wu, «Performance-effective and low-complexity task scheduling for heterogeneous computing», *Transactions on Parallel and Distributed Systems, IEEE,* vol. 13, n° 13, pp. 260-274, 2002.

[14] D.-M. Kwai et B. Parhami, «Periodically regular chordal rings: generality, scalability, and VLSI layout», *Eighth IEEE Symposium on Parallel and Distributed Processing,* pp.148-151, 1996.

[15] M. Saldaña, L. Shannon et P. Chow, «The routability of multiprocessor network topologies in FPGAs», *Proceedings of the 2006 international workshop on System-level interconnect prediction*, pp. 49-56 , 2006.

[16] H. H. Ali et H. El-Rewini, «The Time complexity of Scheduling Interval Orders with Communication Is Polynomial», *Parallel Processing Letters,* vol. 3, pp. 53-58, 1993.

[17] Y.-W. Zhong, J.-G. Yang et H.-N. Qi, «A hybrid genetic algorithm for tasks scheduling in heterogeneous computing systems», *Conference on Machine Learning and Cybernetics, 2004. Proceedings of 2004 International*, pp. 2463-2468, 2004.

[18] R. Sakellariou et H. Zhao, «A hybrid heuristic for DAG scheduling on heterogeneous systems», *18th International Proceedings on Parallel and Distributed Processing Symposium, 2004.*, p. 111, 2004.

[19] X. Tang, K. Li et D. Padua, «Communication contention in APN list scheduling algorithm», *Science in China Series F: Information Sciences,* vol. 52, n° 11, pp. 59-69, 2009.

[20] S. Jin, G. Schiavone et D. Turgut, «A performance study of multiprocessor task scheduling algorithms», *The Journal of Supercomputing,* vol. 43, n° 11, pp. 77-97, 2008.

[21] R. Hwang, M. Gen et H. Katayama, «A comparison of multiprocessor task scheduling algorithms with communication costs», *Computers & Operations Research,* vol. 35, n° 13, pp. 976-993, 2008.

[22] E. Ilavarasan, P. Thambidurai et R. Mahilmannan, «Performance effective task scheduling algorithm for heterogeneous computing system», *The 4th International Symposium on Parallel and Distributed Computing, 2005. ISPDC 2005.*, pp. 28-38. 2005.

[23] I. Ahmad, Y.-K. Kwok et M.-Y. Wu, «Performance comparison of algorithms for static scheduling of DAGs to multiprocessors», *Second Australasian Conference on Parallel and Real-time Systems*, pp. 185-192, 1995.

[24] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen et others, «A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems», *Journal of Parallel and Distributed computing,* vol. 61, n° 16, pp. 810-837, 2001.

[25] Y.-K. Kwok et I. Ahmad, «Static scheduling algorithms for allocating directed task graphs to multiprocessors», *ACM Comput. Surv.,* vol. 31, n° 14, pp. 471, 406, dec 1999.

[26] B. Demiroz et H. R. Topcuoglu, «Static Task Scheduling with a Unified Objective on Time and Resource Domains», *The Computer Journal,* vol. 49, n° 16, pp. 731-743, 2006.

[27] H. Zhao et R. Sakellariou, «An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm», *Euro-Par 2003 Parallel Processing*, Springer, pp. 189-194, 2003.

[28] T. C. Hu, «Parallel Sequencing and Assembly Line Problems», *Operations Research,* vol. 9, n° 16, pp. 841-848, 1961.

[29] Papadimitriou et Yannakakis, «Scheduling Interval-Ordered Tasks», *SICOMP: SIAM Journal on Computing,* vol. 8, pp. 405-409, 1979.

[30] J. E. G. Coffman et R. L. Coffman, «Optimal Scheduling for Two Processor Systems», *Acta Informatica,* vol. 1, pp. 200-213, 1972.

[31] T. L. Adam, K. M. Chandy et J. R. Dickson, «A Comparison of List Schedules for Parallel Processing Systems», *Communications of the ACM,* vol. 17, n° 112, pp. 685-690, dec 1974.

[32] H. Kasahara et S. Narita, «Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing», *IEEE Trans. on Computers,* vol. 33, n° 111, p. 1023, nov 1984.

[33] B. Krautrachue et T. Lewis, «Grain size determination for parallel processing», *IEEE softw.,* vol. 5, n° 11, pp. 23-32, jan 1988.

[34] J. Colin, P. Chretienne et C. P. M., «Scheduling with Small Communication Delays and Task Duplication», *Operations Research,* vol. 39, n° 14,pp. 680-684, jul 1991.

[35] Y.-C. Chung et S. Ranka, «Applications and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors», *SC*, pp. 512-521, 1992.

[36] M. A. Palis, J.-C. Liou et D. S. L. Wei, «Task Clustering and Scheduling for Distributed Memory Parallel Architectures», *IEEE Transactions on Parallel and Distributed Systems,* Vols. 1 sur 2PDS-7, n° 11, pp. 46-55, jan 1996.

[37] A. Girault, H. Kalla et Y. Sorel, Une heuristique d'ordonnancement et de distribution tolérante aux pannes pour systèmes temps-réel embarqués, *Modélisation des Systemes Réactifs, MSR'03*: 145-160, 2003

[38] J. Bannister et K. Trivedi, «Task allocation in fault-tolerant distributed systems», *Acta Informatica,* vol. 20, n° 13, pp. 261-281, 1983.

[39] A. Girault, H. Kalla, M. Sighireanu et Y. Sorel, «An algorithm for automatically obtaining distributed and fault-tolerant static schedules», In : *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, pp. 159, 2003.

[40] J. C. Browne et S. J. Kim, «A General Approach to the Mapping of Parallel Computations upon Multiprocessor Architectures», *Proc. of the 1988 International Conference on Parallel Processing*, University Park, Penn, 1988.

[41] M.-Y. Wu et D. D. Gajski, «Hypertool: A Programming Aid for Message-Passing Systems», *IEEE Transactions on Parallel and Distributed Systems,* Vols. 1 sur 2PDS-1, n° 13, pp. 330-343, jul 1990.

[42] T. Yang et A. Gerasoulis, «Dsc: Scheduling parallel tasks on an unbounded number of processors», *IEEE Transactions on Parallel and Distributed Systems,* vol. 5, pp. 951-967, 1994.

[43] Kim et Yi, «A Two-Pass Scheduling Algorithm for Parallel Programs», *PARCOMP: Parallel Computing,* vol. 20, pp. 869-885, 1994.

[44] C. McGreary et H. Gill, «Automatic Determination of Grain Size for Efficient Parallel Processing», *Communications of the ACM,* vol. 32, pp. 1073-1078,1989.

[45] J. Baxter et J. Patel, «The LAST algorithm- A heuristic-based static task allocation algorithm», *1989 International Conference on Parallel Processing, University Park, PA,* 1989.

[46] JJ. Hwang, YC. Chow, FD. Anger et CY. Lee, «Scheduling Precedence Graphs in Systems with Interprocessor Communication Times», *SICOMP: SIAM Journal on Computing,* vol. 18, pp. 244-257, 1989.

[47] G. C. Sih et E. A. Lee, «A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogenous Processor Architectures», *IEEE Transactions on Parallel and Distributed Systems,* Vols. 1 sur 2PDS-4, n° 12, pp. 175-187, feb 1993.

[48] N. Mehdiratta et K. Ghose, «A Bottom-Up Approach to Task Scheduling in Distributed Memory Multiprocessors», *ICPP*, pp. 151-154, 1994.

[49] Y. Kwok et I. Ahmad, «Benchmarking the task graph scheduling algorithms», *Proc. Int'l Parallel Processing Symp./Symp. on Parallel and Distributed Processing*, pp. 531-537, 1998.

[50] Nasr, Harb et Meghabghab, «Enhanced Simulated Annealing Techniques for Multiprocessor Scheduling», *FLAIRS Conference*, pp. 124-128, 1999.

[51] H. Chen et A. M. Cheng, «Applying Ant Colony Optimization to the Partioned Scheduling Problem for Heterogeneous Multiprocessors», *ACM SIGBED Review* vol. 2, no 2, pp. 11-14, 2005.

[52] M. Ennigrou et K. Ghedira, « Approche Multi-Agents basée sur la Recherche Tabou pour le Job Shop flexible », *RFIA 04: Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes & Intelligence Artificielle*, France, 2004.

[53] W. Hu, J. Song et W. Li, «A New PSO Scheduling Simulation Algorithm Based on an Intelligent Compensation Particle Position Rounding off», *Fourth International Conference on Natural Computation, 2008.* pp. 145-149, *ICNC'08.* 2008.

[54] E. Hou, N. Ansari et H. Ren, «A genetic algorithm for multiprocessor scheduling», *IEEE Transactions on Parallel and Distributed Systems,* vol. 5, n° 12, pp. 113-

120, 1994.

[55] H. Singh et A. Youssef, «Mapping and scheduling heterogeneous task graphs using genetic algorithms», *5th IEEE Heterogeneous Computing Workshop (HCW'96)*, 1996.

[56] A. Wu, H. Yu, S. Jin, K. Lin et G. Schiavone, «An incremental genetic algorithm approach to multiprocessor scheduling», *IEEE Transactions on parallel and distributed systems,* pp. 824-834, 2004.

[57] I. Ahmad et M. Dhodhi, «Multiprocessor scheduling in a genetic paradigm», *Parallel Computing,* vol. 22, n° 13, pp. 395-406, 1996.

[58] P. Shroff, D. Watson, N. Flann et R. Freund, «Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments», *5th Heterogeneous Computing Workshop (HCW'96)*, pp. 98-117, 1996.

[59] L. Wang, H. Siegel et V. Roychowdhury, «A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments», *Proc. Heterogeneous Computing Workshop*, pp. 15-19, 1996.

[60] R. Correa, A. Ferreira et P. Rebreyend, «Integrating list heuristics into genetic algorithms formultiprocessor scheduling», *Eighth IEEE Symposium on Parallel and Distributed Processing, 1996*, pp. 462-469, 1996.

[61] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen et others, «A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems», *Proceedings of the Eighth Heterogeneous Computing Workshop*, pp. 15-29, 1999.

[62] G. Harik et F. Lobo, «A parameter-less genetic algorithm», *GECCO*, vol. 99, Citeseer, pp. 258-267, 1999.

[63] P. Switalski et F. Seredynski, Multiprocessor Task Scheduling Based on GEO Metaheuristic.

[64] P. Switalski et F. Seredynski, «Solving multiprocessor scheduling problem with GEO metaheuristic», *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing-Volume 00*, pp. 1-8, 2009.

[65] P. Switalski et F. Seredynski, «Generalized Extremal Optimization for Solving Multiprocessor Task Scheduling Problem», *Proceedings of the 7th International*

Conference on Simulated Evolution and Learning, pp 161-169, 2008.

[66] I. Shin, A. Easwaran et I. Lee, «Hierarchical scheduling framework for virtual clustering of multiprocessors», *Proceedings of the 20th Euromicro conference on real-time systems*, pp. 181-190., 2008.

[67] A. Coskun, T. Rosing et K. Whisnant, «Temperature aware task scheduling in MPSoCs», *Proceedings of the conference on Design, automation and test in Europe*, pp. 1659-1664, 2007.

[68] K. Stavrou et P. Trancoso, «Thermal-aware scheduling for future chip multiprocessors», *EURASIP Journal on Embedded Systems,* vol. 2007, n° 11, p. 40, 2007.

[69] A. Coskun, T. Rosing, K. Whisnant et K. Gross, «Static and dynamic temperature-aware scheduling for multiprocessor SoCs», *IEEE Transactions on VLSI,* vol. 16, n° 19, pp. 1127-1140, 2008.

[70] S. Sahni, «Scheduling master-slave multiprocessor systems», *IEEE Transactions on Computers,* vol. 45, n° 110, pp. 1195-1199, 1996.

[71] J. Liou et M. Palis, «An efficient task clustering heuristic for scheduling DAGs on multiprocessors», *Workshop on Resource Management, Symposium on Parallel and Distributed Processing*,pp. 152-156, 1996.

[72] I. Ahmad et Y.-K. Kwok, «A new approach to scheduling parallel programs using task duplication», *Conference on Parallel Processing, 1994. ICPP 1994 Volume 2. International*, pp. 47-51, 1994.

[73] S. Gupta, G. Agarwal et V. Kumar, «Task Scheduling in Multiprocessor System Using Genetic Algorithm», *2010 Second International Conference on,Machine Learning and Computing (ICMLC),* pp. 267-271, 2010.

[74] S.-C. Cheng, D.-F. Shiau, Y.-M. Huang et Y.-T. Lin, «Dynamic hard-real-time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints», *Expert Systems with applications,* vol. 36, n° 11, pp. 852-860, 2009.

[75] M. Houshmand, E. Soleymanpour, H. Salami, M. Amerian et H. Deldari, «Efficient scheduling of task graphs to multiprocessors using a combination of modified simulated annealing and list based scheduling», *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*

(IITSI), , pp. 350-354., 2010.

[76] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest et R. Lauwereins, «Energy-aware runtime scheduling for embedded-multiprocessor socs», *IEEE Design & Test of Computers,* pp. 46-58, 2001.

[77] N. Bambha, S. Bhattacharyya, J. Teich et E. Zitzler, «Hybrid global/local search strategies for dynamic voltage scaling in embedded multiprocessors», *Proceedings of the ninth international symposium on Hardware/software codesign*, pp. 243-248., 2001.

[78] G. Varatkar et R. Marculescu, «Communication-aware task scheduling and voltage selection for total systems energy minimization», *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, pp. 510, 2003.

[79] C. Yang, J. Chen et T. Kuo, «An approximation algorithm for energy-efficient scheduling on a chip multiprocessor», *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, pp. 468-473, 2005.

[80] R. Teodorescu et J. Torrellas, «Variation-aware application scheduling and power management for chip multiprocessors», *Proceedings of the 35th International Symposium on Computer Architecture*, pp. 468-473, 2008.

[81] A. Girault, C. Lavarenne, M. Sighireanu et Y. Sorel, «Generation of fault-tolerant static scheduling for real-time distributed embedded systems with multi-point links», *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, p*p. 30125b-30125b, *FTPDS'01.*

[82] B. Kalyanasundaram et K. Pruhs, «Fault-tolerant scheduling», *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 115-124, 1994.

[83] R. Somani et G. Manimaran, «A new fault-tolerant technique for improving schedulability in multiprocessor real-time systems», *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 8, 2001.

[84] R. Capilla, F. Nava et J. Dueas, «Modeling and documenting the evolution of architectural design decisions», *Second Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent, 2007. SHARK/ADI'07: ICSE Workshops 2007.*, pp. 9, 2007.

[85] P. Lago et P. Avgeriou, «First workshop on sharing and reusing architectural knowledge.», *ACM SIGSOFT Software Engineering Notes,* vol. 31, n° 15, pp. 32-

36, 2006.

[86] K. Keutzer et T. Mattson, «A design pattern language for engineering (parallel) software», *Intel Technology Journal,* vol. 13, n° 14, 2010.

[87] C. Alexander, *The timeless way of building,* New York: Oxford University Press, *1979.*

[88] K. Beck et W. Cunningham, *Using pattern languages for object-oriented programs*, 1987, 2006

[89] D. Roberts, R. Johnson et others, «Evolving frameworks: A pattern language for developing object-oriented frameworks», *Pattern languages of program design,* vol. 3, pp. 471-486, 1996.

[90] I. Gorton, «Understanding Software Architecture», *Essential Software Architecture*, Springer, pp. 1-15, 2011.

[91] U. Zdun, «Systematic pattern selection using pattern language grammars and design space analysis», *Software: Practice and Experience,* vol. 37, n° 19, pp. 983-1016, 2007.

[92] N. B. Harrison, P. Avgeriou et U. Zdlin, «Using patterns to capture architectural decisions», *Software, IEEE,* vol. 24, n° 14, pp. 38-45, 2007.

[93] L. Zhu, M. A. Babar et R. Jeffery, «Mining patterns to support software architecture evaluation», *Proceedings Fourth Working IEEE/IFIP Conference on Software Architecture, 2004. WICSA 2004..*, pp. 25-34, 2004.

[94] A. L. Santos et K. Koskimies, «Modular Hot Spots: A Pattern Language for Developing High-Level Framework Reuse Interfaces using Aspects.», *EuroPLOP*, 2008.

[95] J. Borchers, «A pattern approach to interaction design», *AI & Society,* vol. 15, n° 14, pp. 359-376, 2001.

[96] J. Dummler, R. Kunis et G. Runger, «A scheduling toolkit for multiprocessor-task programming with dependencies», *Euro-Par 2007 Parallel Processing*, Springer, 2007, pp. 23-32.

[97] *Systems Engineering for Intelligent Transportation Systems*, 2006-07.

[98] M. Dubois et M. Boukadoum, «Towards an automated framework for task scheduling»,

International Conference on Microelectronics (ICM), 2010, pp. 479-482, 2010.

[99] J.-P. Bordat, « Calcul pratique du treillis de Galois d'une correspondance », *Mathématiques et Sciences humaines,* vol. 96, pp. 31-47, 1986.

[100] S. Ferre, «Incremental concept formation made more efficient by the use of associative concepts », 2002.

[101] M. Dubois et M. Boukadoum, «Association rules learning technique for knowledge mining about scheduling algorithm performance», *New Circuits and Systems Conference (NEWCAS), 2011 IEEE 9th International*, pp. 65-68, 2011.

[102] R. Agrawal, T. Imielinski et A. Swami, «Mining association rules between sets of items in large databases», *ACM SIGMOD Record*, pp. 207-216, 1993.

[103] C. Romero, J. R. Romero, J. M. Luna et S. Ventura, «Mining Rare Association Rules from e-Learning Data.», *EDM*, pp. 171-180. 2010.

[104] J. Han, J. Pei et Y. Yin, «Mining frequent patterns without candidate generation», *ACM SIGMOD Record*, pp. 1-12, 2000.

[105] R. Agrawal, R. Srikant et others, «Fast algorithms for mining association rules», *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pp.487-499, 1994.

[106] M. Dubois et M. Boukadoum, «Rules Maps for Scheduling Algorithm Knowledge», *IEEE International Symposium on Circuits and Systems,* pp. 1728-1731, 2013.

[107] M. Dubois et M. Boukadoum, «Modeling and documentation knowledge extraction for list type scheduling algorithms», *Submitted for publication,* 2013.

[108] M. Dubois et M. Boukadoum, «Rules class approach to scheduling algorithms», *2011 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS),* , pp. 732-735., 2011.

[109] M. Dubois et M. Boukadoum, «A pattern language approach in scheduling algorithm knowledge», *Submitted for publication,* 2013.

[110] S. Mingsheng, S. Shixin et W. Qingxian, «An efficient parallel scheduling algorithm of dependent task graphs», *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003.*, pp. 595-598, 2003.

[111] J. Duato, S. Yalamanchili et L. Ni, Interconnection networks: An engineering approach, Morgan Kaufmann Pub, 2003.

[112] AHMAD, Ishfaq, KWOK, Yu-Kwong, et WU, Min-You. Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors. *Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on*. IEEE, p. 207-213, 1996.

[113] Pang,Yi,Sun, Lifeng,Wen, Jiangtao, Zhang, Fengyan, Hu, Weidong, Feng, Wei, Yang, Shigiang. A framework for heuristic scheduling for parallel processing on multicore architecture: a case study with multiview video coding. *Transactions IEEE on Circuits and Systems for Video Technology*. IEEE, vol 19, numéro 11, p. 1658-1666, 2009.