Universidade Federal Fluminense

MAICON MELO ALVES

An Interference-aware Virtual Machine Placement Strategy for Small-scale HPC Applications in Clouds

NITERÓI

2018

Universidade Federal Fluminense

MAICON MELO ALVES

An Interference-aware Virtual Machine Placement Strategy for Small-scale HPC Applications in Clouds

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Topic Area: Computer Systems.

Advisor:
LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND

NITERÓI

2018

Ficha catalográfica automática - SDC/BEE

A474a Alves, Maicon Melo
An Interference-aware Virtual Machine Placement Strategy for Small-scale HPC Applications in Clouds / Maicon Melo Alves; Lúcia Maria de Assumpção Drummond, orientadora. Niterói, 2018.
124 f.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2018.

1. Computação em nuvem. 2. Computação paralela e distribuída. 3. Sistema de computador virtual. 4. Ciência da computação. 5. Produção intelectual. I. Título II. Drummond,Lúcia Maria de Assumpção, orientadora. III. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

MAICON MELO ALVES

An Interference-aware Virtual Machine Placement Strategy for Small-scale HPC Applications in Clouds

Approved on February 28th, 2018 by:

Prof. Lúcia M. A. Drummond, D.Sc. / IC-UFF (President)

Prof. Yuri Abitbol de Menezes Frota, D.Sc. / IC-UFF

Prof. Simone de Lima Martins, D.Sc. / IC-UFF

Prof. Maria Cristina Silva Boeres, Ph.D. / IC-UFF

Prof. Valmir Carneiro Barbosa, Ph.D. / COPPE-UFRJ

Prof. Philippe Olivier Alexandre Navaux, D.Sc. / UFRGS

Dr. Luiz Rodolpho Rocha Monnerat, D.Sc. / PETROBRAS

Niterói

2018

"All things are difficult before they are easy"

Thomas Fuller

To my greatest accomplishments: my wife, Cristiane, and my daughters, Giovana and Camila.

Acknowledgements

I have been waiting a long time for this moment. The moment to recognize and thank to people who helped me to overcome all obstacles I had faced off in this research.

First and foremost, I would like to thank God for giving me the courage to face all the problems and difficulties encountered in the last four years.

I want to thank my wife, Cristiane and my daughters, Giovana and Camila, for their love and patience, particularly in the moments when I had to dedicate my time and energy to this thesis. Believe me: you are my true treasure.

I also thank my mother, Fátima, my sister, Karoline, and my brother, Johnny. Undoubtedly, you are my safe harbor. Despite all problems, I would like to also dedicate this work to my father, Jorge.

The Ph.D. graduation was probably the most challenging journey I have ever taken in my entire life. Surely, I would not have completed this task without the valuable guidance, support and encouragement of my advisor, Prof. Lúcia Drummond. I am truly honoured to have been guided by one of the most prominent researcher in parallel and distributed computing area. I will always be grateful to her for giving me this incredible opportunity.

I would like to thank Petrobras for all support during this whole process. More specifically, I thank my coordinator, Jamyl, my sector manager, Isac, and my manager, Waldir, for their support and confidence. I also thank my team who holding down the fort while I was away from daily activities. I am really proud to be part of this company which I believe belonging to brazilian folk.

I also thank professors of Computing Institute at UFF whom I had the pleasure to work with or be part of their class: Rosseti, Loana, Satoru and Simone. Especially, I would like to thank Prof. Yuri for assisting us in some parts of this research. In addition, I could not forget to thank Murilo Alves for providing the necessary conditions so that I could conduct this research at UFF/Campos. Last but not least, I thank my colleagues Luan and Rodrigo for their inestimable help on conducting experiments and writing research papers.

Resumo

Em um ambiente de nuvem computacional, aplicações de alto desempenho podem sofrer interferência ao serem executadas em máquinas virtuais que estejam alocadas em uma mesma máquina física. Embora alguns trabalhos tenham proposto estratégias de alocação de máquinas virtuais cientes deste problema, nenhuma dessas estratégias empregou um método adequado para predizer a interferência nem considerou, ao mesmo tempo, tanto a minimização da interferência quanto do número de máquinas físicas ativas na nuvem. Nesta tese, define-se o Problema de Alocação de Máquinas Virtuais ciente da Interferência para Aplicações de Alto Desempenho de Baixa Escalabilidade, um problema que tem a finalidade de minimizar, simultaneamente, (i) a interferência sofrida por aplicações de alto desempenho que estejam sendo executadas em uma mesma máquina física e (ii) o número de máquinas físicas necessárias para alocar essas aplicações na nuvem. Este trabalho apresenta uma formulação matemática para o problema, além de propor uma estratégia baseada na metaheurística Busca Local Iterada para resolvê-lo. Para predizer a interferência, esta estratégia utiliza um modelo quantitativo e multivariado que leva em conta a quantidade e similaridade de acesso aos recursos compartilhados e o número de aplicações co-alocadas. Uma análise experimental, utilizando aplicações reais da área de petróleo e o *benchmark* HPCC, mostraram que o método proposto foi capaz de superar, em termos de redução de interferência, várias heurísticas da literatura. Os resultados reveleram que, mesmo usando o número de máquinas físicas indicados por tais heurísticas, a estratégia proposta conseguiu reduzir o nível de interferência sofrido pelas aplicações alocadas na nuvem.

Palavras-chave: Alocação de Máquinas Virtuais, Interferência entre Aplicações, Computação em Nuvem, Computação de Alto Desempenho.

Abstract

The cross-interference problem may occur when applications are executed in virtual machines placed in a same physical machine. Although many previous works have proposed several different strategies for Virtual Machine Placement, neither of them have employed a suitable method for predicting cross-interference nor have considered the minimization of the number of used physical machines at the same time. In this thesis, we define the Interference-aware Virtual Machine Placement Problem for small-scale HPC applications in Clouds (IVMPP) that tackles both problems by minimizing, at the same time, the cross-interference of small-scale HPC applications, that can share physical machines, and the number of physical machines used to allocate them. We propose a mathematical formulation and a strategy based on the Iterated Local Search framework to solve this problem. Moreover, we also propose a quantitative and multivariate model to predict interference for a set of applications allocated to the same physical machine. Experiments executed in a real scenario, by using applications from the oil and gas industry and the HPCC benchmark suite, showed that our method outperforms several heuristics from the related literature in terms of interference, while using the same number of physical machines.

Keywords: Virtual Machine Placement, Cross-application Interference, Cloud Computing, High Performance Computing.

List of Figures

List of Tables

Acronyms and Abreviations

BF	:	Best Fit;
BFD	:	Best Fit Decreasing;
BP	:	Bin Packing;
DFT	:	Discrete Fourier Transform;
DGEMM	:	Double-precision General Matrix Multiply;
DRAM	:	Dynamic Randmon Access Memory;
FF	:	First Fit;
FFD	:	First Fit Decreasing;
FFT	:	Fast Fourier Transform;
GCC	:	Gnu C Compiler;
GRASP	:	Greedy Randomized Adaptive Search Procedure;
HPC	:	High Performance Computing;
HPCC	:	High Performance Computing Challenge;
HPL	:	High Performance Linpack;
ILS	:	Iterated Local Search;
IVMPP	:	Interference-aware Virtual Machine Placement Problem for HPC applications;
KVM	:	Kernel-based Virtual Machine;
MPI	:	Message Passing Interface;
MRA	:	Multiple Regression Analysis;
MUFITS	:	Multiphase Filtration Transport Simulator;
NUMA	:	Non-uniform Memory Access;
PAPI	:	Performance Application Programming Interface;
PKTM	:	Pre-stack Kirchoff Time Migration;
PM	:	Physical Machine;
PTRANS	:	Parallel Matrix Transpose;
QPI	:	Quick Path Interconnect;
SAR	:	System Activity Report;
SLLC	:	Shared Last Level Cache;

SPE : Society of Petroleum Engineers;
TCO : Total Cost of Ownership;
VBP : Vector Bin Packing;
VM : Virtual Machine;
VMP : Virtual Machine Placement;
VND : Variable Neighborhood Descent;
VNS : Variable Neighborhood Search;
WF : Worst Fit;
WFD : Worst Fit Decreasing;
WSS : Working Set Size;

Glossary

$B_{i,s}$	:	individual access of application i to shared resource s;
$V_{i,j,s}$	:	amount of access of VM V_j to a shared resource s, considering application i;
N_i	:	total number of virtual machines hosting application i;
$T_{s,m}$	:	accumulated access to a shared resource s in a physical machine m;
A_m	:	set of applications allocated to physical machine m;
F_m	:	interference level for applications allocated to physical machine m;
L_{i,A_m}	:	slowdown of application i when co-located with set of applications A_m;
K_{i,A_m}	:	execution time of app. i when executed concurrently with applications A_m;
H_i	:	isolated execution time of application i;
Z	:	sum of interference levels;
M	:	set of available physical machines;
$E_{i,j,s}$	:	similarity factor between applications i and j concerning shared resource s;
$G_{m,s}$	:	global similarity factor for physical machine m regarding shared resource s;
u	:	2-combinations of A_m;
R_m	:	concurrency factor for physical machine m;
n	:	maximum number of applications that can be allocated to the physical machine;
$\overrightarrow{REM}$	:	vector of remaining resources in a physical machine;
$\overrightarrow{REQ}$	:	vector of requested resources to be allocated in a physical machine;
$\overrightarrow{RAF}$	:	vector of remaining resources after allocating requested resources;
θ	:	alignment factor;
τ	:	residual factor;
A	:	set of applications to be allocated in the cloud environment;
Mem	:	total amount of memory available on each physical machine;
Cpu	:	total amount of CPU available on each physical machine;
m_i	:	amount of memory requested by application i;
c_i	:	amount of CPU requested by application i;
γ_Q	:	interference level experienced by subset of applications Q;
ω	:	maximum interference level achieved in the environment;
$X_{i,j}$	:	equal to 1 whether application i is allocated to machine j and 0, otherwise;

Y_j	:	equal to 1 whether physical machine j is being used and 0, otherwise;
α	:	weight to determine the relevance of each objective of mathematical model;
$Z(s)$	:	normalized sum of interference levels for the solution s;
$M(s)$	:	normalized number of physical machines used in solution s;
$E_{CPU}(s)$	:	percentage of amount of CPU exceeded in solution s;
$E_{MEM}(s)$	:	percentage of amount of main memory exceeded in solution s;
$\mathbb{S}$	:	set of all possible solutions for a given instance of IVMPP;
λ	:	penalization for exceeding amount of CPU and main memory;

Contents

Chapter 1

Introduction

High Performance Computing (HPC) is broadly used for solving complex problems in several scientific fields such as cosmology, molecular dynamics, quantum chemistry, climate and human genetics [41]. HPC enables scientists to extrapolate their knowledge beyond theoretical and experimental fields in order to understand and optimize known scenarios, or even for predicting natural phenomena like storms, earthquakes and tornadoes. Thus, high computational power delivered by HPC systems is essential to enhance human knowledge by helping scientist to answer the most fundamental questions concerning the universe.

HPC has also been employed to solve engineering or business problems like the ones faced by petroleum and automotive industries. Specifically in petroleum industry, HPC is used in processes responsible for improving oil and gas production in a new or developed field [59] [40] or to explore areas where petroleum can be found and recovered [11] [30]. In addition, HPC has become indispensable to execute Big Data analysis in the large and complex amount of data continuously gathered by companies [80]. From data analysis results, these companies can get insight into customer behavior so that they can recommend products and services according to its needs and desires [16].

Besides that, HPC has been frequently used to tackle combinatorial optimization problems such as packing [26], vehicle routing [83], resource allocation and scheduling [31]. In this sort of problems, an optimal solution must be identified from a finite set of feasible solutions. Due to the high number of possible solutions, HPC is employed to accelerate search methods like heuristics and meta-heuristics in such a way that a near-optimal solution can be found in a reasonable time.

Typically, HPC applications are executed in dedicated datacenters. In general, these

environments are equipped with performance-oriented machines, and low latency and high bandwidth networks like Infiniband [50] [82] [18]. More specifically, low latency is particularly critical for synchronous and tightly-coupled HPC applications [49] since a single delay in communication layer can affect their entire execution. Moreover, in these environments, HPC applications are singly executed in a given physical machine in order to avoid concurrent access to shared and non-sliceable resources [43]. For all these reasons, a specialized and dedicated infrastructure is still nowadays the main option to run HPC applications.

However, in the past few years, Cloud Computing has emerged as a promising alternative to execute these applications. This new computational paradigm brings some attractive advantages when compared with a dedicated infrastructure, such as rapid provisioning of resources and significant reduction in operating costs related to energy, software licence and hardware obsolescence [13] [34] [90]. In order to leverage the adoption of clouds for running HPC applications, some initiatives, such as UberCloud, have provided a free and experimental HPC on-demand service, where users can discuss their experiences by using such environment for executing their HPC applications [38].

Despite these advantages, some challenges must be overcome to bridge the gap between performance provided by a dedicated infrastructure and the one supplied by clouds. Overhead introduced by virtualized layer [27] and hardware heterogeneity [42], for example, affect negatively the performance of HPC applications when executed in clouds. In addition, the absence of a high-performance network can prevent synchronous and tightly-coupled HPC applications from being satisfactorily executed in this environment [13] [70] [44] [50].

Besides these problems, the performance of HPC applications can be particularly affected by resource sharing policies usually adopted by cloud providers. In general, one physical server can host many virtual machines holding distinct applications [69] [28]. Although virtualization layer provides a reasonable level of resource isolation [48], some shared resources, like cache and main memory, cannot be sliced over all applications running in the virtual machines. As a consequence, these co-located applications can experience mutual interference, resulting in performance degradation [43] [51] [53].

In face of the cross-application interference problem, some works, such as [43], [52], [96], [21], [51], [28], and [78], proposed Virtual Machine Placement (VMP) strategies to avoid or, at least, reduce the effect that interference imposes in co-located HPC applications. Some of those works just used a static matrix of interference or proposed a naive

procedure to determine it. Other works, even proposing more general methods to determine interference, considered just one kind of shared resource, the Shared Last Level Cache (SLLC), or adopted an approach that evaluates only general and subjective characteristics of HPC applications. However, as discussed in Chapter 3, all of these methods are not suitable for determining the interference because this problem is directly related to the amount and similarity of concurrent access to SLLC, DRAM (Dynamic Random Access Memory) and virtual network, and to the number of co-located applications.

Moreover, the vast majority of those works proposed VMP strategies that neglected an important issue on clouds: the minimization of the number of used physical machines. Those works did not evaluate that goal because it is, actually, the opposite of reducing interference. Indeed, to minimize the number of physical machines, a VMP strategy should consolidate all applications in a small number of physical machines, increasing consequently the interference. On the other hand, the interference could be alleviated by spreading out applications in the set of available physical machines, what would rise the number of used physical machines. However, minimizing the number of used physical machines is essential to reduce operational costs related to power consumption and cooling systems [25].

Concerning the number of physical machines, [43] and [51] proposed VMP strategies that aims to minimize the number of used physical machines also considering the interference problem. However, those works neither treated both of those problems together nor sought for the effective minimization of interference. In [43], the VMP strategy treats both problems separately by prioritizing one or another objective depending on the class of HPC application. In [51], the authors proposed a solution which attempts to minimize the number of physical machines while keeping interference within bounds. Although conflicting, the minimization of interference and the number of used physical machines can be evaluated together by using a multi-objective approach. Thus, a VMP strategy, aware of interference, can allocate virtual machines in such a way that a better trade-off between minimizing interference and number of used machines can be achieved.

1.1 Objective

In this thesis, we define the Interference-aware Virtual Machine Placement Problem for Small-scale HPC Applications in Clouds (IVMPP), a multi-objective problem that aims to minimize, at the same time, (i) the interference experienced by small-scale HPC ap-

plications executed in a same physical machine and (ii) the number of physical machines used to allocate them.

IVMPP targets small-scale HPC applications, i.e., applications that can be entirely placed in a physical machine, because the network interconnection still remains as the major performance limiting factor for running medium and high scale HPC applications in clouds [89] [35] [81]. Even in specific HPC clouds as the Amazon EC2 Compute Cluster [66], the network layer prevents tightly-coupled HPC applications, which performs a lot of synchronous and intensive communication, from being satisfactorily executed [50]. On the other hand, clouds offer a reasonable service performance for small-scale HPC applications because such applications can avoid the performance penalty usually imposed by the physical network [49] [43] [36] [47]. At last, notice that, with the recent increase of the number of cores in modern physical machines[1], small-scale HPC applications can scale up to many cores when fully allocated to a single physical machine.

Firstly, we define formally the IVMPP by introducing a mathematical formulation, and then we propose a strategy based on the Iterated Local Search (ILS) framework to solve it. That proposed strategy, called IVMPP_ILS, is basically comprised of three phases. At first, the IVMPP_ILS executes a greedy constructive method to generate a high quality initial solution. After creating this initial solution, the IVMPP_ILS executes a VND (Variable Neighborhood Descent) procedure to improve the current solution until reaching the local optima. Next, the algorithm applies a perturbation in the current local minimum in order to escape from the local optima. Furthermore, at the end of each ILS iteration, the IVMPP_ILS evokes the constructive method to create a new solution to be submitted to the ILS. This multistart approach provides the necessary diversification to overcome local minimum towards global optimum [64].

In order to predict interference, the IVMPP_ILS uses a multivariate and quantitative prediction model also proposed in this work. This proposed interference prediction model takes into account (i) the amount of simultaneous accesses to shared resources, (ii) the similarity between the applications access profiles, and (iii) the number of co-located applications. More specifically, our proposed model considers the effect that SLLC, DRAM and virtual network concurrent accesses produce in cross-application interference. Those three resources are considered particularly critical because (i) SLLC and DRAM are shared among cores of a processor [93] [12] and, (ii) virtual network, although not being a hard-

[1]Recently, Intel® launched processor Xeon Platinum 8180M with 28 cores. Thus, a physical server that supports two of these processors, such as Gigabyte® MD61-SC2, provides 56 cores in total. Similarly, the server Supermicro® AS-1023US-TR4 provides 64 cores with two AMD® EPYC 7601. In addition, the SW26010 processor, a homegrown Chinese manycore chip, has 260 cores in total.

ware resource, is emulated by the hypervisor, shared by all virtual machines.

The cross-interference prediction model was validated by using two real-life HPC applications frequently used in the petroleum industry, namely the Multiphase Filtration Transport Simulator (MUFITS) [72] and the Pre-stack Kirchhoff Time Migration (PKTM) [14], and applications provided by the High Performance Computing Challenge (HPCC) benchmark [23]. Those applications were executed with different instances as input and the results showed that our model predicted cross-application interference with maximum and median errors of 13.88% and 3.59%, respectively. Besides that, the prediction error was less than 10% in 93% of all tested cases.

Concerning the minimization of the number of used physical machines, we adopted in our VMP strategy an approach similar to the one employed by [43]. This approach is based on the Vector Bin Packing problem, a variant of the classical Bin Packing problem, and aims to balance the use of physical resources by considering both the size and shape of the exploitable volume of resources available in the cloud environment [67].

Our proposed VMP strategy was evaluated by using a set of instances created from the aforementioned validation workload. Results achieved by our strategy were compared with the ones obtained by the most widely used heuristics to minimize the number of machines in clouds. The obtained results indicate that, even using the same number of physical machines given by those heuristics, our strategy was able to reduce the level of interference experienced by co-located applications up to 41%.

1.2 Contributions

The main contributions of this work are the following:

1. A formal definition and the corresponding mathematical formulation for the Interference-aware Virtual Machine Placement Problem for Small-scale HPC Applications in Clouds (IVMPP). This problem aims to minimize, simultaneously, the interference experienced by applications allocated to the same physical machine and the number of physical machines used to allocate them in the cloud environment.

2. A strategy based on the Iterated Local Search (ILS) framework to solve the IVMPP. Besides being composed of the most common ILS components like perturbation and local search heuristics, the proposed strategy also employs a multistart mechanism. In this mechanism, new solutions, generated from a greedy constructive method,

are submitted to the ILS at the end of each iteration. This approach enables our strategy to cover a larger area of the set of feasible solutions.

3. A synthetic application template which allows to create a set of applications that perform distinct access levels to shared resources. Such applications are created by varying properly the input parameters of the proposed template. A synthetic application based on this template executes two well-defined phases commonly present in HPC applications.

4. An empirical evaluation presenting the multivariate and quantitative nature of the cross-application interference problem. This experimental analysis was carried out by using a set of synthetic applications specially created to investigate the cross-application interference problem. This synthetic workload, which was generated from our proposed application template, is comprised of applications that put distinct access pressure on shared resources.

5. A multivariate and quantitative model able to predict cross-application interference level by considering (i) the amount of concurrent accesses to SLLC, DRAM and virtual network, (ii) the similarity between the amount of applications accesses, and (iii) the number of co-located applications.

6. An experimental analysis, executed in a real scenario, of both the prediction model and the VMP strategy, by using a real reservoir petroleum simulator, a seismic migration algorithm, and applications from a well-known HPC benchmark.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related works and discusses the main contributions of our proposal when compared with them. Chapter 3 describes the proposed multivariate and quantitative model for predicting interference, showing its building process and validation. Chapter 4 presents the criterion used by the proposed VMP strategy to minimize the number of used physical machines. In Chapter 5, we formally define the IVMPP and introduce the proposed VMP strategy to solve it. Finally, conclusions and directions for future work are presented in Chapter 6.

Chapter 2

Related Work

At first, this chapter introduces works which deals with the cross-application interference problem in Section 2.1. Then, we describe in Section 2.2 papers that solve the Virtual Machine Placement (VMP) problem considering the interference issue. Finally, in Section 2.3, we present papers that consider both the minimizing of interference and number of used physical machines to solve the VMP problem.

2.1 Cross-application Interference Problem

Mury *et al.* [68] argued that cross-application interference could be determined by adopting a classification of applications. Such qualitative approach is based on the *Thirteen Dwarfs* which classifies applications according to computational methods usually adopted in scientific computing. Such classification is not suitable for determining interference because, as will be described in Section 3.4.3, two applications belonging to a same class can present distinct interference levels. Besides that, these results also pointed out that a same application can present distinct interference levels when solving instances with different sizes. Actually, experiments conducted in [43] also showed this same behavior when applications EP (Embarrassingly Parallel) and LU (Lower-Upper Gauss-Seidel solver)[1] presented distinct interference levels when solving instances of different sizes.

Chi *et al.* [29] evaluated experimentally the degree at which the performance of a guest virtual machine degrades under different combinations of background workloads. These experiments were carried out by using single-threaded applications like basic Linux commands (cat, gzip, gpg, bzip2), memory tests (cachebench) and games (gnugo). These simple applications are quite different from HPC applications because the latter typically

[1]Applications EP, LU, CG, IS and FT are provided by the Nas Parallel Benchmark (NPB).

uses the virtual network for communication. As that paper tested only single-threaded applications, the effect of virtual network access contention in cross-application interference was not properly assessed. However, as will be described in Section 3.2.3, the level of interference experienced by co-located HPC applications can vary significantly depending on the amount of accumulated access to virtual network. Therefore, the level of dispute over this shared resource should also be considered when predicting interference for this sort of application.

Koh *et al.* [53], similarly to [29], also used single-threaded applications, such as gzip2 and grep, to evaluate the extent to which the performance of co-located applications is affected when running in a virtualized environment. They conducted interference experiments by allocating two applications to the same physical machine in order to investigate the cause of this problem. As stated before, those single-threaded applications are not suitable for properly assessing the interference problem for HPC applications. This happens because single-threaded applications do not allow to evaluate the concurrent access to virtual network, one of the causes of cross-interference problem. Moreover, by considering just two co-located applications, that work ignored the fact that the interference level tends to increase with the rising of the number of co-located applications, as will be shown in experiments described in Section 3.5.

Rameshan *et al.* [78] proposed a procedure to prevent latency sensitive applications, such as video streaming and web services, from being adversely affected by batch applications when co-located in a same physical machine. In such proposal, latency sensitive applications calls the proposed procedure whenever they are under interference caused by batch applications. The procedure, based on the information collected when it is called, predicts the next period of interference. From this prediction, the procedure throttles the batch application before it imposes interference to latency applications one more time. Thus, that work just proposed a way to work around the interference experienced by latency sensitive applications by monitoring the conditions that lead to occurrence of this problem. So, neither the cause of the interference problem was investigated nor a solution to determine interference experienced by HPC applications was proposed.

Unlike those works, our proposed model is able to predict interference for HPC applications by taking into account the amount of concurrent access to virtual network and to other shared resources like SLLC and DRAM.

2.2 Virtual Machine Placement Aware of Interference

Basto [21] argued that a VMP strategy, aware of interference, could improve the performance of applications when executed in clouds. To validate this argument, the author modified the simulator Simgrid so that it simulated the cross-application interference effect by using a static interference matrix. This interference matrix, that was based on results obtained from other papers, is composed of values related to the interference effect for three types of applications: cpu-bound, memory-bound and i/o-bound. So, by consulting the type of application in this matrix, Simgrid was able to simulate the corresponding interference level for the set of co-located applications. After modifying Simgrid, the author presented a new version of the OpenStack scheduler which selects the physical machine with the highest processor frequency and the lowest expected interference to host incoming virtual machines. This expected interference, which is also calculated from this static interference matrix, indicates the degree of interference currently present in the physical host. Since the work used only interference values observed in previous papers, it did not conduct any investigation to figure out the cause of the cross-interference problem. Besides not proposing any method to determine interference, that work did not also evaluate its approach in a real physical machine.

Yokoyama *et al.* [96] proposed an interference-aware VMP strategy to allocate virtual machines in a private HPC cloud. Their proposed strategy seeks for the maximization of application throughput, i.e., it aims to execute, in a given time interval, the maximum number of applications in this environment. In order to reach this goal, their proposal tries to reduce applications execution times by alleviating the interference effect. The VMP strategy reduces the interference effect by co-locating applications with complementary access profiles according to a static interference matrix. This interference matrix was acquired from interference experiments accomplished with benchmark applications such as HPL (High Performance Linpack) and PARPACBench. So, by using the same set of applications earlier considered in the interference experiments, the authors evaluated the VMP strategy and showed that the throughput can really be increased through an interference-aware co-location. The authors stated, however, that the obtained interference values were influenced by factors not investigated. So, they only verified the hypothesis that reducing interference is really essential to rise the performance of HPC applications when executed in clouds.

Jin *et al.* [52] classified applications in three SLLC access classes, called (i) cache-pollution, (ii) cache-sensitive and (iii) cache-friendly. Then, they proposed a VMP strat-

egy based on the co-locating of HPC applications with compatible SLLC access profiles. This work claimed that cache-pollution applications should be preferably co-located with cache-friendly applications rather than being co-located with cache-sensitive ones. However, through some practical experiments they showed that the approach may fail. More specifically, EP and IS (Integer Sort), classified as cache-friendly, experienced distinct interference levels when co-located with CG (Conjugate Gradient), categorized as a cache-pollution application. Besides that, although both CG and FT (Fast Fourier Transform) were classified as cache-pollution applications, CG did not present interference when co-located with itself, whereas FT experienced interference when co-located with CG. These results indicate that a qualitative approach based on SLLC access pattern is not suitable for determining cross-application interference precisely.

Chen *et al.* [28] introduced the cache contention aware VMP problem. This problem aims to minimize performance degradation of virtual machines by reducing SLLC access contention in a physical host. Besides presenting a formal definition of the problem, the authors also proposed a heuristic to solve it. Such heuristic tries to minimize the total cache contention degree by avoiding to co-locate virtual machines with intensive cache consumption, i.e., virtual machines that use a large space of the shared cache. To estimate the total cache contention degree, this work proposed an algorithm based on the concept of stack distance which allows to capture the temporal reuse behavior of an application when executed in a system with a fully or set-associative cache. Although this paper proposed a quantitative solution to determine interference, the proposed method deems SLLC access contention as being the unique cause of cross-application interference. However, as will be discussed in Section 3.2.3, the level of interference experienced by co-located applications is affected by concurrent access to virtual network and DRAM, as well. In addition, their proposed VMP strategy is application-agnostic. This means that the VMP strategy treats each virtual machine as a separate placement problem, that is, it disregards the relation between virtual machines and applications running in the top of them [41]. As a consequence, the VMP strategy may allocate a set of virtual machines, holding the same application, in distinct physical machines. Due to the absence of a high-performance network, this approach can worsen the performance of an HPC application by spreading out the corresponding virtual machines across the set of physical hosts available in the cloud environment.

The works above either introduced only the interference problem or proposed methods which are insufficiently accurate to predict cross-application interference. Moreover, all of them overlooked the importance of minimizing the number of physical machines used

to allocate applications in clouds. Unlike those works, the VMP strategy proposed in this paper aims to minimize, simultaneously, the interference experienced by co-located applications and the number of physical machines needed to allocate them.

2.3 Virtual Machine Placement Aware of Interference and Usage of Physical Machines

Jersak *et al.* [51] modified some well-known placement heuristics to consider not only the number of used machines but also the cross-application interference. Thus, the authors adapted the heuristics FFD (First Fit Decreasing), BFD (Best Fit Decreasing) and WFD (Worst Fit Decreasing) to not overcome an interference threshold when allocating virtual machines to physical hosts. In order to predict interference, the authors devised a simple interference model used as a proof of concept with their heuristics. In such model, the level of interference is defined as a function of the number of virtual machines co-executing in a physical machine. So, the model considers that the higher the number of virtual machines, the higher the interference level is. However, in our work we show that for the same number of virtual machines, the interference can vary drastically. This happens because interference is related to the amount of access to shared resources and not to the number of virtual machines being co-executed in a host. Additionally, those heuristics aim only not to overcome a predefined interference threshold, and do not seek for minimizing the interference.

Gupta *et al.* [43] proposed a VMP strategy that attempts either to avoid interference or to minimize the number of used machines, depending on the class of HPC applications. Tightly-coupled synchronous HPC applications, for example, are always allocated to a dedicated physical machine, i.e., a machine that will host only a single application at a time. On the other hand, for the remaining classes of HPC applications, the VMP strategy seeks for the minimization of the number of physical machines, though also observing an acceptable interference criteria. This interference criteria is based on a maximum SLLC accumulated access limit which is defined from experimental analysis. Authors claimed that this SLLC accumulated threshold guarantee an acceptable level of interference for applications allocated to the same physical machine. Although this work proposed a quantitative strategy to alleviate the interference problem, only one shared resource, SLLC, was considered. However, as will be presented in Section 3.2.3, our experiments showed that other shared resources such as virtual network and DRAM can also influence interference and, consequently, should be systematically evaluated. Moreover,

their proposed VMP strategy treats, separately, the minimization of interference and the number of used physical machines because, depending on the HPC class, it focuses on solving one or another problem. Furthermore, this VMP strategy did not minimize, in fact, the interference since it is just concerned about not trespassing a given interference limit.

Both of papers proposed VMP strategies that, despite being interference-aware, treated, separately, the problems of minimizing interference and the number of used physical machines. Actually, by considering only an interference limit, none of them really attempted to minimize the level of interference that arises in a cloud environment. Note that, even within a maximum interference limit, there could be allocation arrangements that would result in lower interference levels. Moreover, those works employed, in their VMP strategies, a naive or incomplete method to determine interference.

In addition, their methods were based on benchmark applications results. However, this kind of application may be not suitable to investigate the interference problem since the amount of access to shared resources cannot be controlled. Consequently, experiments conducted with those applications do not allow to evaluate, systematically, the relation between distinct access levels to shared resources and resulting interference.

Unlike those works, we propose a VMP strategy which aims to minimize, simultaneously, the level of interference experienced by co-located applications and the number of physical machines, by using a quantitative and multivariate prediction model which evaluates concurrent access to three shared and non-sliceable resources.

Chapter 3

Cross-application Interference Prediction Model

In order to minimize the interference level, the proposed VMP strategy uses an interference prediction model that was built upon a synthetic interference dataset. This dataset is comprised of distinct levels of access to shared resources and the corresponding interference level experienced by the set of co-located applications. From this interference dataset, the model was created by using the Multiple Regression Analysis, a multivariate statistical technique.

Firstly, we describe in Section 3.1 some basic assumptions and definitions used throughout this thesis. Next, in Section 3.2, we present an interference dataset which was specially created to investigate the interference problem. Then, we describe the cross-application interference model created upon that interference dataset in Section 3.3. The validation of the proposed model is discussed in Section 3.4, while an extended prediction model is presented in Section 3.5. At last, in Section 3.6, we discuss the precision of the extended model by comparing it with a model exclusively generated from real experimental results.

3.1 Basic Definitions and Assumptions

As stated in Chapter 1, our work focuses on small-scale HPC applications, that is, HPC applications that are entirely allocated to a single physical machine. Thus, in this work, we assumed that virtual machines holding one application are not allocated to more than one physical machine. In other words, the allocation of one application to a physical machine is equivalent to the allocation of all virtual machines, holding that application, to the same physical machine.

In this work, we considered three shared and non-sliceable resources when investigating interference: (i) SLLC, (ii) DRAM and (iii) virtual network. We evaluated SLLC because concurrent access to this shared resource was pointed out in previous papers, as earlier described in Chapter 2, as one of the major causes of interference problem. Since SLLC access contention can impose a high interference to co-located applications, we also investigated concurrent access to DRAM, another critical component of memory subsystem. In addition, we also investigated the virtual network access contention because this shared resources is commonly used by small-scale HPC applications running in a same physical machine.

We now define formally the amount of individual and accumulated access of applications to shared resources, besides describing metrics used to measure the cross-application interference in the cloud environment.

The *individual access* (B) of an application i to a shared resource s is defined as the sum of access of all virtual machines, holding this application, to s, where N_i is the total number of virtual machines hosting i and $V_{i,j,s}$ is the amount of access of virtual machine V_j, hosting application i, to a shared resource s:

$$B_{i,s} = \sum_{j=1}^{N_i} V_{i,j,s} \tag{3.1}$$

In the context of this work, the amount of access to SLLC and DRAM are measured in terms of millions of references per second (MR/s), while the access to virtual network is expressed as the amount of megabytes transmitted per second (MB/s).

From *individual access*, we defined the *accumulated access* (T) as the sum of all individual access (B) to a shared resource s performed by applications co-located in m, where A_m is the set of applications allocated to the physical machine m:

$$T_{s,m} = \sum_{\forall i \in A_m} B_{i,s} \tag{3.2}$$

This accumulated access represents the total pressure put by applications in a given shared resource.

In this work, the negative impact experienced by co-located applications is measured in terms of the *interference level* (F):

$$F_m = \frac{\sum_{\forall i \in A_m} L_{i,A_m}}{|A_m|} \tag{3.3}$$

So, the interference level experienced by the set of applications allocated to m is calculated as the average slowdown (L) of applications allocated to this physical machine.

Particularly, in this thesis, the *slowdown* is expressed as the percentage of additional time spent by one application when it is concurrently executed with other ones. Formally, the slowdown (L) of one application i is equal to the ratio of the execution time achieved by this application when executed concurrently with other ones (K_{i,A_m}) to the time when executed by itself (H_i) minus 1:

$$L_{i,A_m} = \frac{K_{i,A_m}}{H_i} - 1 \tag{3.4}$$

For example, suppose that the execution times of two applications, namely A1 and A2, when executed in a dedicated physical machine, were equal to 60 and 80 seconds, respectively. Suppose also that both applications, when concurrently executed in that physical machine, spent 100 seconds. In this case, the slowdown of applications A1 and A2 would be, respectively, 67% and 25%. This percentage represents how much additional time these applications needed to complete their executions when allocated to the same physical machine. Thus, the interference level between these applications would be 46%, which means that these two applications would experience, in average, 46% of mutual interference when allocated to the same physical machine.

From the interference level, we define the *sum of interference levels* which is used to assess the overal interference of cloud environment. Formally, the sum of interference levels (Z) is equal to the sum of interference levels calculated for each physical machine of the cloud environment (M):

$$Z = \sum_{\forall m \in M} F_m \tag{3.5}$$

At last, we assumed that the interference level is equal to zero in cases where just one application is allocated to a physical machine. This assumption is derived from the fact that this isolated application does not contend for any shared resource.

3.2 Generating a Cross-application Interference Dataset

As mentioned before, we built our proposed interference prediction model from an interference dataset that was specially created to investigate the cross-application interference problem.

In order to create that interference dataset, several co-locating experiments were conducted by using applications with distinct access burden. Those applications were generated from a synthetic application template, originally presented in this work.

3.2.1 Generating Synthetic Applications

Access contention to shared resources is the main cause of interference experienced by applications allocated to the same physical machine. To study cross-application interference, the vast majority of previous papers employed real applications provided by traditional HPC benchmarks. However, a real application does not allow to control the number of accesses to each shared resource, and consequently, to evaluate systematically the relation between concurrent accesses and the resulting interference.

Thus, we propose an *application template* from which synthetic applications with distinct access levels are created. From this template, we can create an application which puts a high pressure to SLLC, while keeping a low access level to virtual network, for example. Thus, a set of synthetic applications created from that template allows to observe interference in face of different levels of accesses to SLLC, DRAM and virtual network.

In order to represent the usual behavior of HPC applications [85], the proposed synthetic application template presents, alternately, two distinct and well-defined phases. The first one, called *Computation Phase*, represents the phase at which the application performs tasks involving calculation and data movement. The other one, namely *Communication Phase*, is the phase where the application exchanges information among computing pairs.

The proposed application template is shown in Algorithm 1. Firstly, the synthetic application executes the *Main Loop* (lines 1 to 13) whose total number of iterations is controlled by parameter *AppIter*. This parameter adjusts the execution time of the application. Next, the synthetic application executes *Computation Phase Loop* (lines 2 to 9) at which it performs the *Computation Phase* with the number of iterations defined

by *Comp*. This *Computation Phase* is based on the benchmark STREAM [10] which is widely used to measure the performance of memory subsystems [92]. In order to measure sustainable memory bandwidth, this benchmark executes four simple vector kernels, as presented in Table 3.1. Because SUM presents the highest access memory ratio, it was chosen to be included in the proposed template.

Algorithm 1 Synthetic application template

```
   Input parameters: AppIter, Comp, AccessStep, WSSCtrl, CompInt, DtAmount,
   Comm
   /* Main Loop*/
1: for x=1 to AppIter do
      /* Computation Phase Loop*/
2:    for y=1 to Comp do
         /* Memory Access Loop*/
3:       for i=1 to WSSCtrl step AccessStep do
4:          A[i] = B[i] + C[i];
            /*Compute-intensive Loop*/
5:          for k=1 to CompInt do
6:             T = SquareRoot(T);
7:          end for
8:       end for
9:    end for
      /* Communication Phase Loop*/
10:   for z=1 to Comm do
11:      All-to-All-Communication(D,DtAmount);
12:   end for
13: end for
```

The SUM operation is executed by the inner loop *Memory Access Loop* (lines 3 to 8) which is controlled by two input parameters, *WSSCtrl* and *AccessStep*. The first one defines the sizes of vectors A, B and C, and is indirectly used to determine application's Working Set Size (WSS) [54]. A small WSS usually increases application's cache hit ratio because all data needed by it in a given time interval can be entirely loaded in cache. On the other hand, when the application has a WSS greater than cache capacity, its cache hit ratio decreases because all data is fetched from main memory. Thus, WSS can be used to control the cache hit ratio and, consequently, control the number of DRAM references per second also.

The second parameter of *Memory Access Loop*, *AccessStep*, controls the step at which the vector elements are accessed. Thus, when *AccessStep* is equal to 1, all elements of vectors A, B and C are accessed consecutively, resulting in a high cache hit ratio. Otherwise, when *AccessStep* is set to a high value, more data is fetched from main memory,

Name	Operation	Bytes per Iteration
COPY	`a[i] = b[i]`	16
SCALE	`a[i] = b[i]*q`	16
SUM	`a[i] = b[i] + c[i]`	24
TRIAD	`a[i] = b[i] + c[i]*q`	24

Table 3.1: STREAM kernels

resulting in performance degradation. In other words, this parameter provides another way to control the cache hit ratio and to manipulate the number of DRAM references per second.

Thus, the number of DRAM references per second and application's cache hit ratio can be controlled by performing a fine tuning of both parameters *WSSCtrl* and *AccessStep*. When the application presents a high cache hit ratio, DRAM receives few references per second because data is already available in SLLC. Likewise, the number of memory references increases when application presents a low cache hit ratio.

Besides controlling DRAM access, these parameters allow to handle the number of SLLC references per second as well. When the application presents a low cache hit ratio, the number of SLLC references per second decreases. This happens because, before accessing new data, the previous referenced data, not found in SLLC, has to be fetched from DRAM. Consequently, the application's data access rate is reduced, decreasing also the number of SLLC references per second. On the other hand, when the application presents a high cache hit ratio, the number of SLLC references per second increases. Because most data is rapidly fetched from SLLC, application increases the number of memory access requests per second.

After performing the SUM operation (line 4), the synthetic application executes *Compute-intensive Loop* which repeatedly calculates the square root of variable T (line 6). This loop, whose number of iterations is defined by *CompInt*, makes the application more or less compute-intensive. Note that when *CompInt* is set to a high value, the number of memory references decreases. This happens because variable T, being frequently referenced, is kept stored in the first cache level (cache L1), preventing memory subsystem lower levels from being accessed. As a result, SLLC and DRAM references per second decreases. Thus, together with *WSSCtrl* and *AccessStep*, *CompInt* is also used to manipulate the number of DRAM and SLLC references per second.

After *Computing Phase Loop* execution, the synthetic application performs *Communication Phase* by executing *Communication Phase Loop* (lines 10 to 12) whose number of

iterations is determined by the input parameter *Comm*. This phase is based on MPBench benchmark [5], a tool commonly used to evaluate distributed memory systems based on MPI (Message Passing Interface) [46]. From all MPI operations tested by this benchmark, `MPI_Alltoall` was particularly interesting for this work because it is widely used in scientific applications. When using this collective operation, all application's processes send and receive to/from each other the same amount of data [87].

For each *Comunnication Phase Loop* iteration, the application executes *All-to-All-Communication()* function (line 11) that employs `MPI_Alltoall` of a vector D whose size is defined by the input parameter *DtAmount*. So, this input parameter is used to handle the number of bytes transmitted in the virtual network.

Synthetic applications with distinct access profiles can be generated by varying properly these aforementioned input parameters, whose descriptions are summarized in Table 3.2.

Input Parameter	**Description**
AppIter	Application's total number of iterations
Comp	Total number of iterations of *Computation Phase*
Comm	Total number of iterations of *Communication Phase*
WSSCtrl	Working set size
AccessStep	Step which vector elements are accessed
CompInt	Total number of iterations of *Compute-intensive Loop*
DtAmount	Amount of data exchanged among processes

Table 3.2: Application template input parameters

Unlike adopting real applications, with this proposed synthetic application template, several applications that put distinct pressure levels to SLLC, DRAM and virtual network can be generated, providing a proper way to investigate systematically the relation between number of accesses and cross-application interference.

3.2.2 Used Synthetic Applications

In this section, we present the set of synthetic applications generated from the previously presented application template. Applications with distinct amounts of individual accesses were generated, considering three target access levels for each of the three shared resources. The amount of individual accesses to each shared resource is expressed by distinct metrics, such as number of references to memory per second or transmitted bytes per second, and the range of those values are different. To treat those access rates jointly, we normalized

those values in an interval between 0.0 and 1.0, where score 1.0 represents the highest possible access rates achieved by an application based on the proposed template, and score 0.0 represents no access. These scores, in our work, represent different access levels to the shared resources. Then, we created applications with *high*, *medium* and *low* access levels to SLLC, DRAM and virtual network, where the high access level corresponds, in our proposal, to the highest access rate to each shared resource, and medium and low access levels correspond to 50% and 10% of this high access rate, respectively.

To generate those applications with these distinct access levels, we varied the input parameters of the application template and monitored the resulted access rates to each shared resource by using the following monitoring tools: PAPI (Performance Application Programming Interface) [9], OProfile [7] and SAR (System Activity Report) [76][1].

We executed this set of synthetic applications in a Itautec MX214 server whose configuration details are described in Table 3.3. As illustrated in Figure 3.1, this server is equipped with two NUMA (Non-Uniform Memory Access) nodes interconnected by a QPI (Quick Path Interconnect) of 6.4 GT/s. Each NUMA node has 24GB of DRAM memory and a Intel Xeon X5675 3.07GHz processor with six cores sharing a 12MB SLLC unit. Moreover, the virtual environment was provided by KVM (Kernel-based Virtual Machine) hypervisor running on top of Ubuntu Server.

Furthermore, we considered that a single virtual machine has one core and 4GB of main memory. By assigning just one core to each virtual machine, we guarantee that all communication is performed over the virtual network. Thus, we can stress this shared resource to evaluate its influence in the interference experienced by co-located applications.

In our proposal, each application uses six of those virtual machines. We used CPU affinity to deploy half of the virtual machines of the application in each NUMA node, i.e., three virtual machines were deployed in "NUMA Node #1", while the others were deployed in "NUMA Node #2", in a dedicated machine. In this scenario, accesses to shared resources, specifically SLLC and DRAM, are balanced over two NUMA nodes. Moreover, that configuration will be helpful to evaluate cross-application interference as will be discussed in the next section.

The set of the generated synthetic applications and the corresponding access profiles are presented in Table 3.4, whereas the template input parameters, chosen to generate each application, are described in Table 3.5. All applications execute the same number of

[1]Remark that, besides Oprofile and PAPI, other monitoring tools such as Perf [8] can also be used to profile applications by collecting the information available in hardware performance counters.

Model	Itautec MX214
CPU	2x Intel Xeon X5675 3.07 GHz
DRAM	48 GB DDR3 1333 MHz
Disk	5.8 TB SATA 3 GB/s
QPI	6,4 GT/s
Operating System	Ubuntu 15.04
Kernel	3.19.0-15
Hypervisor	KVM
Hardware Emulation	Qemu 2.2.0

Table 3.3: Configuration of the machine used in experiments

iterations (*AppIter* = 25) and parameters *Comp* and *Comm* were set to ensure that they spent approximately the same amount of time executing *Computation* and *Communication Phases*. Moreover, all scores were rounded to one decimal place. This explains, for example, why applications S1 and S7, although having presented distinct absolute values, were classified in the same SLLC score.

	Absolute Value			**Individual Score**		
App.	**SLLC**	**DRAM**	**NET**	**SLLC**	**DRAM**	**NET**
S1	1635	4	300	1.0	0.0	0.1
S2	851	61	324	0.5	0.1	0.1
S3	239	41	312	0.1	0.1	0.1
S4	444	444	318	0.3	1.0	0.1
S5	224	224	324	0.1	0.5	0.1
S6	797	240	318	0.5	0.5	0.1
S7	1597	18	2892	1.0	0.0	1.0
S8	890	43	2810	0.5	0.1	1.0
S9	220	49	2910	0.1	0.1	1.0
S10	438	438	2832	0.3	1.0	1.0
S11	214	214	2892	0.1	0.5	1.0
S12	817	241	2838	0.5	0.5	1.0
S13	1575	22	1392	1.0	0.0	0.5
S14	890	52	1362	0.5	0.1	0.5
S15	228	49	1335	0.1	0.1	0.5
S16	438	438	1375	0.3	1.0	0.5
S17	221	221	1404	0.1	0.5	0.5
S18	824	239	1380	0.5	0.5	0.5

Table 3.4: Generated synthetic applications and the corresponding access profiles when executed in a dedicated machine

Applications S1, S7 and S13 achieved the highest SLLC number of references per second. In order to reach this high SLLC individual access, we adjusted the input parameters to ensure that all memory references were directly satisfied by SLLC, which resulted in a

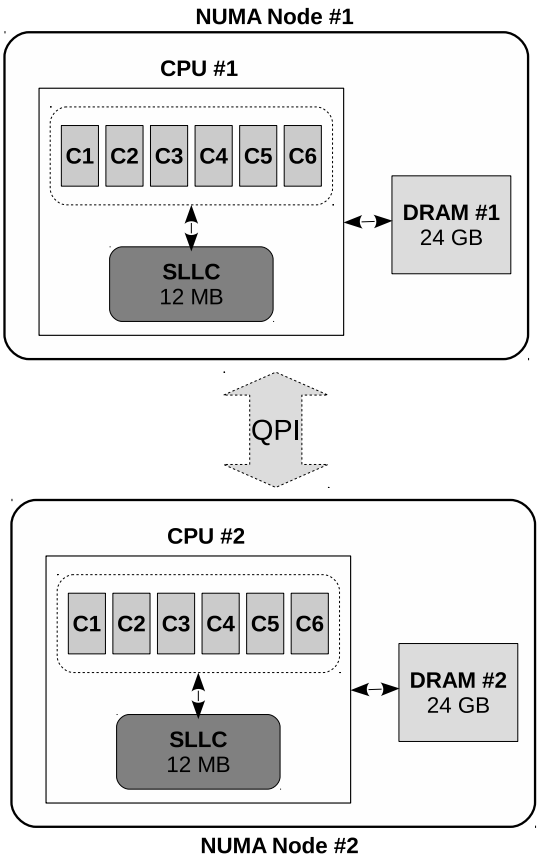


Figure 3.1: NUMA nodes and processors used in our experiments

0.0 DRAM score. Thus, although score 0.0 was not considered as one of the three target access levels, there is no way to achieve the highest number of SLLC references per second without reducing drastically the number of accesses to DRAM.

On the other hand, the number of memory references satisfied by SLLC has to decrease to rise the number of DRAM references per second. To achieve a high DRAM individual access, all memory references should result in accesses to main memory, i.e., the SLLC hit ratio must be close to 0%. However, even in that case, the number of SLLC references per second is not equal to zero, because all references to DRAM are also treated by SLLC. This explains why applications S4, S10 and S16, which achieved the highest number of DRAM references per second, exhibited a SLLC score equal to 0.3, though this score was not considered as one of the target access levels.

Thus, concerning the memory subsystem, we were not able to generate all possible combinations involving the three access levels. A high number of individual accesses to SLLC implies in a low number of accesses to DRAM. As a consequence, it is not possible to generate an application where both SLLC and DRAM scores are equal to 1.0 or an application which puts, simultaneously, a high and medium pressure on SLLC and DRAM,

	Parameters						
App.	**AppIter**	**Comp**	**Comm**	**WSSCtrl**	**AccessStep**	**CompInt**	**DtAmount**
S1	25	120000	5200	7000	512	0	22600
S2	25	90000	5200	9000	1024	6	22600
S3	25	40000	5200	11500	2048	22	22600
S4	25	7500	5200	30000	512	0	22600
S5	25	2700	5200	39000	512	21	22600
S6	25	20000	5200	11800	256	2	22600
S7	25	120000	1500	7000	512	0	749568
S8	25	90000	1500	9000	1024	6	749568
S9	25	40000	1500	11500	2048	22	749568
S10	25	7500	1500	30000	512	0	749568
S11	25	2700	1500	39000	512	21	749568
S12	25	20000	1500	11800	256	2	749568
S13	25	120000	150000	7000	512	0	150000
S14	25	90000	150000	9000	1024	6	150000
S15	25	40000	150000	11500	2048	22	150000
S16	25	7500	150000	30000	512	0	150000
S17	25	2700	150000	39000	512	21	150000
S18	25	20000	150000	11800	256	2	150000

Table 3.5: Template input parameters chosen to create synthetic applications

for example.

At last, concerning virtual network, the highest amount of transmitted bytes was achieved by increasing input parameter *DtAmount* up to reaching the maximum amount of data that the hypervisor is able to handle at the same time. We varied *DtAmount* to find out the virtual network saturation threshold. When this limit is exceeded, the amount of bytes transmitted per second decreases, regardless of increasing *DtAmount*.

Although we did not generate all possible combinations of applications, we were able to create a synthetic workload with distinct computational burden. As can be seen in Figure 3.2, this set of synthetic applications comprises a wide range of access profiles, varying from low access applications, such as S3 and S5, to the ones that put, simultaneously, a high pressure on two shared resources such as S7 and S10. We claim that this broad range of access profiles is suitable for conducting a deeper evaluation of the cross-application interference problem.

3.2.3 Measuring Cross-applications Interference

In this section, we present experiments to determine cross-applications interference. The previously presented synthetic applications were executed in a two-by-two fashion to ob-

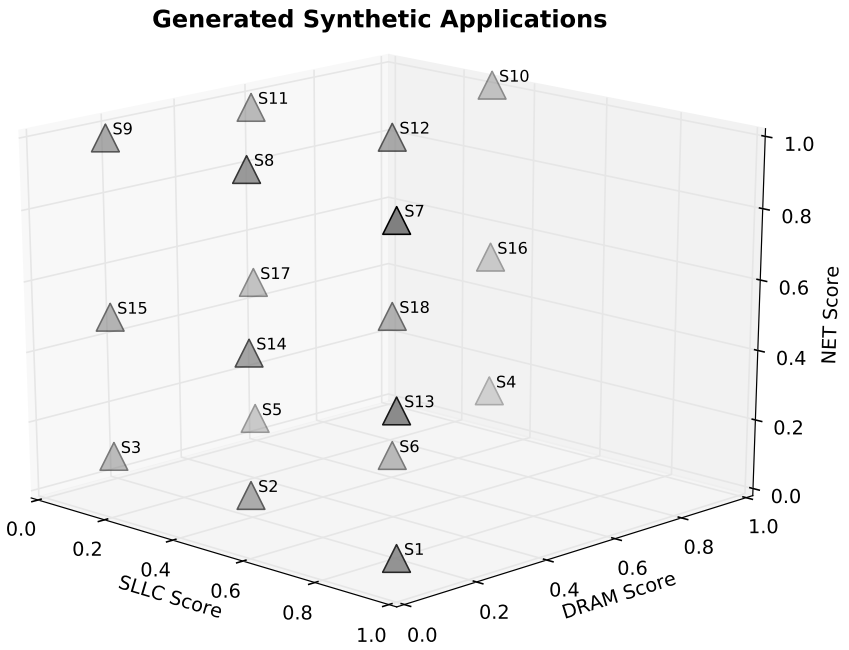


Figure 3.2: Synthetic applications with distinct access profiles

tain the resulting interference level in several cases. Because each synthetic application used half of available resources (memory and CPU), we were able to co-locate two of those applications in the physical machine, without exceeding the available resources in the system. That allocation represents a realistic scenario, usually found in clouds, where all resources available in a physical machine are fully allocated to maximize resource utilization, i.e., to minimize the number of used physical machines in the cloud environment [84].

The generated synthetic applications do not present exactly the same execution time, so to keep concurrency among co-located applications until the end of the experiment, the smaller execution time application was re-started automatically as many times as necessary to cover the entire execution of the longer application. We adopted such approach to fairly measure the interference experienced by both applications, regardless of their execution times.

The overall interference results are summarized in Figure 3.3. Around 54% of the total concurrent executions (93 occurrences) achieved an interference level less than 50%, while 37% of co-locations (63 occurrences) experienced interference levels between 50% and 100%. Besides that, in around 9% of all cases (15 occurrences) co-location applications reached interference levels greater than 100%. These results presented a coefficient of

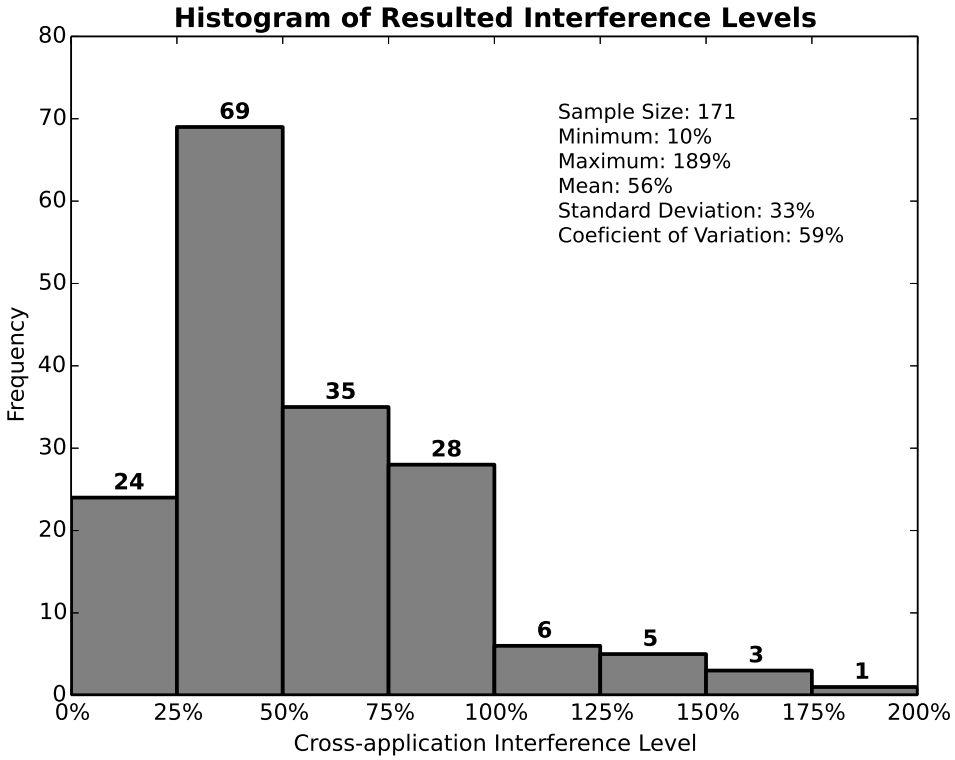


Figure 3.3: Frequency of cross-applications interference levels

variation[2] close to 59% which allows to assert that these synthetic experiments comprised a large range of interference levels.

A deeper analysis accomplished in these results revealed that there is a correlation between interference level and SLLC accumulated score. As can be seen in Figure 3.4, interference level tends to increase as SLLC accumulated score rises. Indeed, the Pearson's correlation coefficient between SLLC accumulated score and interference level is around 0.76, indicating a strong, positive and linear relationship between these both variables. Thus, this observation corroborates the hypothesis that the amount of accesses to shared resources can really influence cross-application interference.

In addition, these experiments allowed to confirm that mutual access to other shared resources besides SLLC can also impact the interference level. Consider, for example, SLLC accumulated score equal to 0.40, in Figure 3.4, it may occur in cases with distinct interference levels. In other words, although SLLC accumulated access presents a strong correlation with interference level, there is, at least, another factor influencing it.

Actually, the interference level increases as concurrent access to virtual network also

[2]Also known as relative standard deviation, the coefficient of variation is defined as the ratio of the standard deviation to the mean.

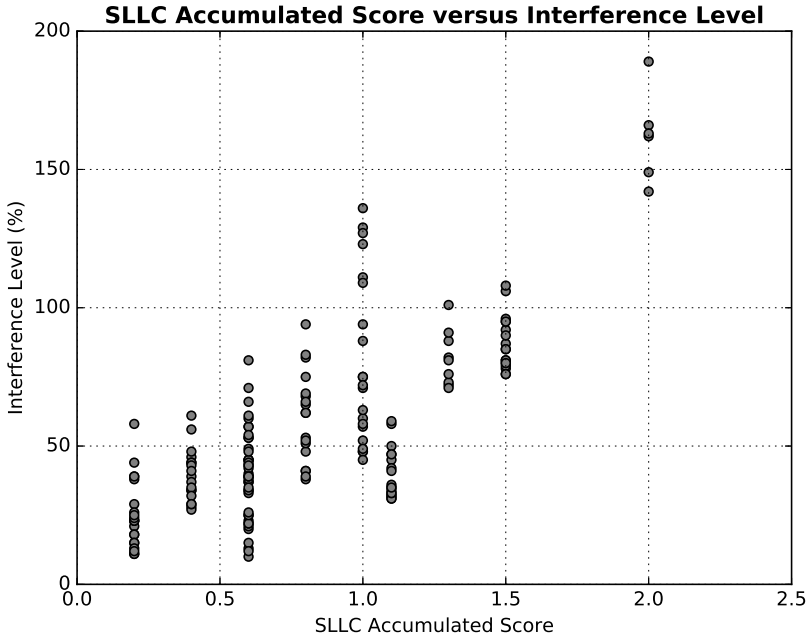


Figure 3.4: SLLC accumulated score versus interference level

does. As illustrated in Figure 3.5, when virtual network accumulated score is equal to 0.2 and 2.0, the corresponding interference levels are around 28% and 60%, respectively. For the same SLLC accumulated score, the cross-application interference levels vary more than 30% depending on the amount of access to virtual network.

Similarly, results revealed that concurrent access to DRAM can also impose a negative impact in interference level. As highlighted in Table 3.6, the interference level can change substantially depending on the accumulated access to DRAM. For instance, application S2, when allocated with itself, achieved an interference level equal to 71.12%. On the other hand, application S6, even presenting the same SLLC and virtual network access profiles than S2, experienced an interference level around 58% higher than the latter. This same behaviour can be observed for applications S8 and S12. Thus, besides SLLC and virtual network, the accumulated access to DRAM can also affect the level of interference experienced by applications that share a common physical machine.

In addition, some co-locations, even though present almost the same amount of accumulated access to the three shared resources, reached interference levels varied by more than 45%. In the subset of interference results, listed in Table 3.7, for example, the co-location “S14xS8” experienced an interference level 46% higher than the co-location “S15xS7”, although both have the same virtual network accumulated score, 1.5, and differ slightly about DRAM and SLLC accumulated scores. Applications S15 and S7 present

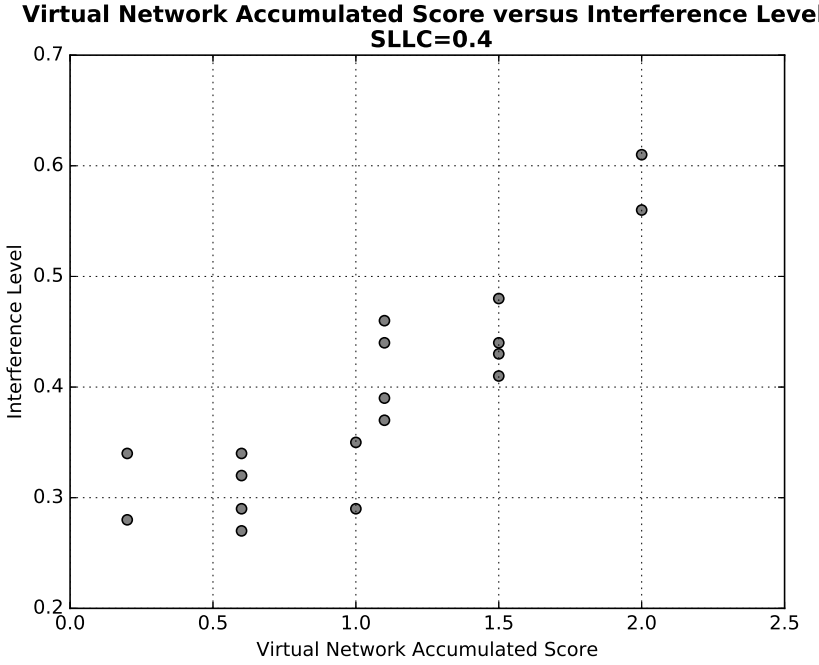


Figure 3.5: Virtual network accumulated score versus interference level when SLLC was equal to 0.40

Co-execution	Accumulated Score			Interference Level
	SLLC	DRAM	NET	
S2 x S2	1.0	0.2	0.2	71.12%
S6 x S6	1.0	1.0	0.2	129.28%
S8 x S8	1.0	0.2	2.0	94.02%
S12 x S12	1.0	1.0	2.0	136.09%

Table 3.6: Subset of experiments showing the influence of DRAM accumulated score in interference levels

distinct SLLC individual access values, the former presents a lower number of SLLC individual access than the latter. This explains why this co-location does not present a high interference level, even achieving a high SLLC accumulated access. On the other hand, the co-location "S14xS8", although presents a SLLC accumulated access similar to co-location "S15xS7", achieved a high interference level because both applications, which present similar SLLC individual access, evenly competed for SLLC. As can be seen in Table 3.7, this also happened in co-locations that present virtual network accumulated scores equal to 0.6 and 0.2.

So, besides accumulated access to shared resources, the similarity between the amounts of application's individual access has a direct impact in the interference level experienced by applications when allocated to a same physical machine. Indeed, this justifies the difference between the interference levels achieved by co-locations listed in Table 3.7.

Co-execution	Accumulated Score			Interference Level
	SLLC	DRAM	NET	
S1 x S3	1.1	0.1	0.2	34.10%
S2 x S2	1.0	0.2	0.2	71.12%
S13 x S3	1.1	0.1	0.6	31.51%
S14 x S2	1.0	0.2	0.6	71.83%
S15 x S7	1.0	0.2	1.5	41.67%
S14 x S8	1.1	0.1	1.5	87.97%

Table 3.7: Results of accumulated score and interference levels for a subset of experiments

In order to measure the level of similarity between two applications, we define the *similarity factor* (E):

$$E_{i,j,s} = 1 - |B_{i,s} - B_{j,s}| \tag{3.6}$$

So, the *similarity factor* of two applications regarding to a shared resource s is calculated as the difference between 1 (highest individual access score) and the absolute value resultant from the difference between the amount of *individual accesses* (B) of applications i and j to a shared resource s.

Note that, because the individual accesses (B) of applications are normalized between 0 and 1, the similarity factor will fall in this same interval. A similarity factor close to 1 indicates a high level of similarity between applications, while values close to 0 mean that the applications present distinct access profiles concerning a shared resource. In other words, the higher the similarity factor is, the higher the level of similarity between two applications will be. For example, concerning SLLC, the similarity factors for co-locations "S14xS8" and "S15xS7" are equal to 1.0 and 0.1, respectively. These values point out that S14 and S8 present a higher level of similarity than the one presented by S14 and S8. Indeed, as previously discussed, the applications S14 and S8 evenly compete for SLLC, while S15 and S7 put distinct access pressure on this shared resource.

As similarity factor is used to assess similarity between pairs of applications, we devised an additional metric to measure the overall level of similarity for all applications placed in the same physical machine. Then, we define the *global similarity factor* (G) as follows:

$$G_{s,m} = \frac{\sum\limits_{\forall i,j \in A_m / i \neq j} E_{i,j,s}}{u} \tag{3.7}$$

In short, the global similarity factor is equal to the average of all similarity factors, concerning a shared resource s, calculated for each pair of applications allocated to the physical machine m, where u is the number of 2-combinations of A_m.

Results of the above described experiments showed that the cross-application interference is influenced by the following factors:

- amount of simultaneous accesses to SLLC;
- amount of simultaneous accesses to virtual network;
- amount of simultaneous accesses to DRAM;
- similarity between the amounts of applications' individual accesses to each shared resource;

As presented above, the cross-application interference problem is influenced by four different factors, which reveals its multivariate nature. Because a multivariate statistical technique allows to examine the relationship among multiple variables, in this work we adopted this method to create a model for predicting the interference level experienced by co-located applications.

3.3 A Multivariate and Quantitative Cross-application Interference Prediction Model

In this section, we describe the proposed prediction model that was created by using the Multiple Regression Analysis technique. In Section 3.3.1, we briefly introduce this statistical method, while in Section 3.3.2 we describe how it was applied to build our prediction model.

3.3.1 Multiple Regression Analysis - Main Concepts

Multiple Regression Analysis (MRA) is a multivariate statistical technique that allows to explain the relationship between one dependent variable and, at least, two independent variables. This technique is used to create a model, called statistical variable, able to predict the value of the dependent variable from the known values of independent variables [88].

Basically, MRA comprises four macro steps as illustrated in flowchart of Figure 3.6. Firstly, in *Variables Selection* step, the most likely variables to explain the behavior of the response variable are selected. Although some statistical techniques such as matrix of correlation can provide some insights about which variables must be chosen, this process is mainly guided by the researcher's knowledge about the problem [45] [71].

After that, the *Model Estimation* step is executed, where the terms of the model are determined and their coefficients are automatically estimated by using the Least Squares Method. Next, the *Model Evaluation* step is executed to evaluate the goodness-of-fit level, i.e., how satisfactorily the estimated model fits to the data used in the building process. Besides that, the statistical significance of regression and coefficients are also assessed. In case that the estimated model does not present a satisfactory goodness-of-fit level or a desired statistical significance level, a new model must be estimated by executing, again, the *Model Estimation* step. Otherwise, the estimated model is ready to be validated in the *Model Validation* step by using an "unseen" dataset, i.e, a dataset not previously used in the process of model estimation [45] [71] [65] [91].

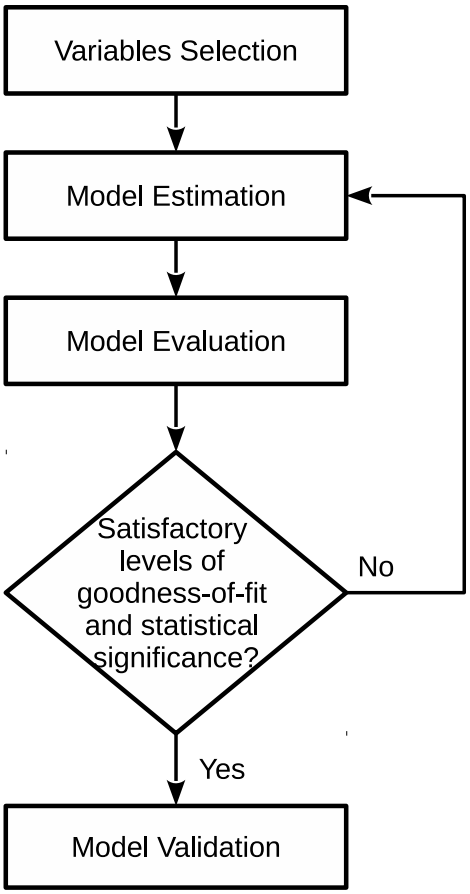


Figure 3.6: Main steps of Multiple Regression Analysis

Although non specialized programs such as Excel and Matlab can be used for building

a model from MRA, this process is usually accomplished by using commercial statistical packages such as Eviews [1], SPSS [3], and Minitab [4], or free software ones like Gretl [20] or R [77]. In general, such specific tools provide a full multivariate analysis toolbox that allows to conduct a deep analysis on the estimated model.

3.3.2 Building the Interference Prediction Model

By using Minitab version 17.1.0, we followed the aforementioned steps to build our proposed model. At first, we executed the *Variables Selection* step to identify which variables should be selected as independent ones. As the synthetic dataset was generated specifically to investigate interference, the selection step was straightforward. We determined the accumulated scores and global similarity factors of the three shared resources as independent variables and cross-application interference level as the dependent one. We aimed to generate a model able to predict the interference level from the known values of accumulated scores and global similarity factors.

After the *Variable Selection* step, we repeatedly executed both the *Model Estimation* and *Model Evaluation* steps until reaching a parsimonious model with satisfactory levels of goodness-of-fit and statistical significance. A parsimonious model is able to perform better out-of-sample predictions because, due to its simplicity, it is not usually overfitted to the sample [91].

Our multivariate and quantitative model for predicting interference is described as follows:

$$\begin{aligned} F_m =& (0.7498 * T_{SLLC,m} * G_{SLLC,m}) + \\ & (0.1598 * T_{NET,m} * G_{NET,m}) + \\ & (0.1456 * T_{DRAM,m} * G_{DRAM,m} * T_{SLLC,m}) \end{aligned} \tag{3.8}$$

As can be seen in the prediction model, the global similarity factors, G_{SLLC}, G_{DRAM} and G_{NET}, were employed to weight the influence that accumulated accesses, T_{SLLC}, T_{DRAM} and T_{NET}, impose in the interference. Moreover, the total amount of access to DRAM was weighted by SLLC accumulated access since all accesses to DRAM are firstly treated in SLLC.

The coefficients calculated for each term, namely 0.7498, 0.1598 and 0.1456, indicate that the interference is more influenced by simultaneous and accumulated access to SLLC,

represented in the model by the term $T_{SLLC,m}*G_{SLLC,m}$. However, as previously discussed, the access to virtual network ($T_{NET} * G_{NET}$) and DRAM ($T_{DRAM} * G_{DRAM,m} * T_{SLLC,m}$), that together presents a coefficient very close to 0.31, can also affect interference. Thus, depending on the value of the SLLC accumulated score, the interference can also be primarily determined by accumulated accesses to DRAM and virtual network.

That model presented an Adjusted Coefficient of Regression, that is Adjusted R-squared (R^2-*adj*), around 0.912, which means that 91.2% of variance can be explained through that estimated model. In other words, high R^2-*adj* indicates that the model is suitable for the used dataset and gives accurate predictions about the dependent variable.

In addition, we assessed statistical significance of the regression model and coefficients of each term by applying the F hypothesis test. At a level of significance of 0.05, test F revealed that regression and its coefficients can be considered as being statistically significant since resulted *p-values* were smaller than the level of significance. These results indicate that each coefficient is a meaningful addition to prediction model because changes in the independent variable value are related to changes in the dependent variable.

Moreover, an analysis accomplished on residuals showed that the estimated model did not violate any of the MRA basic assumptions. Thus, the residuals presented (i) linearity, (ii) homocedasticity[3] and (iii) normal distribution.

3.4 Experimental Tests and Results

In this section, we describe the experimental tests accomplished to assess the quality of our proposed prediction model. In Section 3.4.1, we describe the workload used for conducting experimental tests. In Section 3.4.2, we present predictions made by using our model, while in Section 3.4.3 we evaluate how precisely these predictions match to the interference levels achieved in real experiments.

3.4.1 Description of the Used Real Applications

In order to evaluate the quality of our prediction model, we carried out experiments by using two real applications used in petroleum industry, MUFITS and PKTM, and applications from the High Performance Computing Challenge benchmark (HPCC).

[3]In the context of regression analysis, this term refers to the circumstance in which the variance of residuals across values predicted for the dependent variable is constant.

MUFITS [6] is employed by petroleum engineers to study the behavior of petroleum reservoir over time [72]. From simulation results, they can make inferences about future conditions of the reservoir in order to maximize oil and gas production in a new or developed field [13]. Basically, the simulator employs partial differential equations to describe the multiphase fluid flow (oil, water and gas) inside a porous reservoir rock [73]. Reservoir simulation is one of the most expensive computational problems faced by petroleum industry since a single simulation can take days, even weeks, to finish. Computational complexity of this problem arises from the high spatial heterogeneity of multi-scale porous media [59].

PKTM is a seismic migration method that provides a subsurface image from earth. This migrated image, that is generated from a raw seismic section, can reveal geological structures where oil and gas can be detected and further recovered. A raw seismic section must be migrated because the acquisition process can produce a subsurface image that originally does not represent the real geological structure found in the target area. PKTM is one of the most used migration methods in industry due to its simplicity, efficiency, feasibility and I/O flexibility. Similarly to reservoir simulator, the seismic migration process is also computationally expensive and a single execution can take several hours to finish, even when executed in supercomputers [94] [14].

Besides using both applications, we tested applications from HPCC [2], a widely adopted benchmark to evaluate HPC systems [97]. Although HPCC provides seven kernels in total, we just used the ones that represent real HPC applications or operations commonly employed in scientific computing. A brief description of these four kernels follows [60] [32].

- HPL (High Performance Linpack): solves a dense linear system of equations by applying the Lower-Upper Factorization Method with partial row pivoting. This application, that is usually employed to measure sustained floating point rate of HPC systems, is the basis of evaluation for the Top 500 list[4].
- DGEMM (Double-precision General Matrix Multiply): performs a double precision real matrix-matrix multiplication by using a standard multiply method. Even not being a complex real application, this kernel represents one of the most common operation performed in scientific computing, the matrix-matrix multiplication.
- PTRANS (Parallel Matrix Transpose): performs a parallel matrix transpose. As

[4]https://www.top500.org/

their pairs of processors communicate with each other simultaneously, this application is a useful test to evaluate the total communications capacity of the network.

- FFT (Fast Fourier Transform): computes a Discrete Fourier Transform (DFT) of a very large one-dimensional complex data vector and is often used to measure floating point rate execution of HPC systems.

We considered, for each application, distinct instances to certify that our proposal is able to predict interference, regardless of the size and characteristics of the instance being solved. In MUFITS, we considered instances usually adopted in the related literature. The first one, labeled here as "I1", considers a simulation of CO^2 injection in the Johansen formation by using a real-scale geological model of the formation. The other one, labeled here as "I2", is related to the 10th SPE (Society of Petroleum Engineers) Comparative Study. Both instances are available on the MUFITS Web site [6].

Concerning PKTM, we used synthetic and real seismic sections, labeled here as "I1" and "I2", respectively. The synthetic seismic section, and its respective velocity model, were created by using a synthetic seismograph. This sort of seismic section, such as the Marmousi and SEG/EAGE models, are commonly used for evaluating geophysical solutions, techniques, and algorithms [56] [57] [24] [79] [95], because they faithfully represent geological features found in real seismic sections. Even so, we also used a real seismic section composed of data collected from a real seismic data acquisition. This data was acquired in Brazil, but, for confidentiality reasons, we cannot inform the exact localization where this acquisition took place.

For HPCC, we considered instances whose details are described in Table 3.8. Note that, specifically for PTRANS and DGEMM, we created a small instance, called I3, to be executed with only one virtual machine. Instances for HPCC were created by adjusting parameters "#N", "N" and "NB" which correspond, respectively, to the number of problems, the size of the problem treated by the application and the size of the block. Remark that each input parameter has a specific meaning for each application. For example, in case of DGEMM, input parameter "N" is used to set the dimension of matrices to be multiplied, while this same parameter, in case of FFT, determines the size of vector of real numbers to be transformed to the frequency domain. More detailed information about these input parameters can be found on the HPCC Web site [2].

Application	Instance	HPCC Parameters		
		#N	N	NB
HPL	I1	1	18000	80
	I2	1	15000	80
DGEMM	I1	1	3000	80
	I2	1	18000	80
	I3	1	1000	80
PTRANS	I1	1	24000	80
	I2	5	500	80
	I3	1	18000	80
FFT	I1	1	65000	10
	I2	1	40000	10

Table 3.8: Description of HPCC applications instances

3.4.2 Predicting Interference with the Proposed Model

In order to assess the prediction model in distinct scenarios, we considered four co-location schemes: A, B, C and D. In schemes A and B, all applications, solving both I1 and I2, were co-located in a two-by-two and three-by-three fashions, respectively. In scheme C, we co-located all applications, except PKTM, in a six-by-six fashion, but solving just I1. At last, in D we co-located applications PTRANS and DGEMM in a twelve-by-twelve fashion when solving I3. In addition, each virtual machine used in all co-location schemes has the same configuration as described in Section 3.2.2. So, an application running with 4 virtual machines would use 4 CPU and 16 GB of main memory in total, for example. Moreover, as performed in synthetic experiments, half of virtual machines allocated to each application was deployed to a NUMA node.

As presented before, the prediction model relies on the amount of individual access to shared resources to estimate the cross-application interference. So, to test our proposed model, we firstly executed each of those applications in a dedicated physical machine to acquire their corresponding access profiles to the three shared resources. This profiling was executed in the server previously described in Section 3.2.2.

Individual access scores achieved by each application are described in Table 3.9. FFT achieved the highest access rate to virtual network, while PTRANS, when solving I1, imposed the highest pressure to SLLC and DRAM. Moreover, as DGEMM and PKTM are embarrassingly parallel applications, they presented a low access level to virtual network. It is worth mentioning that some applications decreased their access rates to shared resources when executing with a lower number of virtual machines. This is expected since the same application, when executed with a lower number of processes, decreases, usually,

the amount of individual access to shared resources.

App. Label	Application	Instance	Virtual Machines	Individual Score		
				SLLC	DRAM	NET
MUFITS.I1.P6	MUFITS	I1	6	0.053	0.127	0.004
MUFITS.I2.P6	MUFITS	I2	6	0.030	0.000	0.008
MUFITS.I1.P4	MUFITS	I1	4	0.049	0.075	0.003
MUFITS.I2.P4	MUFITS	I2	4	0.024	0.000	0.006
MUFITS.I1.P2	MUFITS	I1	2	0.016	0.027	0.000
HPL.I1.P6	HPL	I1	6	0.026	0.062	0.015
HPL.I2.P6	HPL	I2	6	0.028	0.057	0.019
HPL.I1.P4	HPL	I1	4	0.018	0.041	0.014
HPL.I2.P4	HPL	I2	4	0.012	0.038	0.011
HPL.I1.P2	HPL	I1	2	0.008	0.011	0.005
DGEMM.I1.P6	DGEMM	I1	6	0.017	0.021	0.000
DGEMM.I2.P6	DGEMM	I2	6	0.010	0.024	0.000
DGEMM.I1.P4	DGEMM	I1	4	0.011	0.023	0.000
DGEMM.I2.P4	DGEMM	I2	4	0.007	0.016	0.000
DGEMM.I1.P2	DGEMM	I1	2	0.004	0.009	0.000
DGEMM.I3.P1	DGEMM	I3	1	0.003	0.005	0.000
PTRANS.I1.P6	PTRANS	I1	6	0.183	0.214	0.322
PTRANS.I2.P6	PTRANS	I2	6	0.018	0.000	0.111
PTRANS.I1.P4	PTRANS	I1	4	0.136	0.091	0.194
PTRANS.I2.P4	PTRANS	I2	4	0.015	0.000	0.038
PTRANS.I1.P2	PTRANS	I1	2	0.054	0.041	0.189
PTRANS.I3.P1	PTRANS	I3	1	0.050	0.029	0.000
FFT.I1.P4	FFT	I1	4	0.070	0.164	0.517
FFT.I2.P4	FFT	I2	4	0.068	0.169	0.493
FFT.I1.P2	FFT	I1	2	0.041	0.086	0.306
PKTM.I1.P6	PKTM	I1	6	0.004	0.000	0.001
PKTM.I2.P6	PKTM	I2	6	0.001	0.000	0.001
PKTM.I1.P4	PKTM	I1	4	0.003	0.000	0.001
PKTM.I2.P4	PKTM	I2	4	0.001	0.000	0.001

Table 3.9: SLLC, DRAM and virtual network individual scores of real applications executions

Considering the individual profile of each application, we applied our model to predict what would be the interference experienced by those applications if they were co-located according to the aforementioned schemes. Prediction results point out that the minimum and maximum predicted interference levels would be equal to 0.20% and 49.60%, respectively. The minimum interference level was predicted for co-location "PKTM.I2.P6 x PKTM.I2.P6", while the maximum one was estimated for FFT.I1.P2 when co-located with itself in a six-by-six fashion. As expected, the lowest and highest interference levels were predicted for co-locations that involved, respectively, applications with low and high access rates to SLLC, DRAM and virtual network. In other words, our model indicated

that PKTM, DGEMM and HPL would suffer a low cross-interference, while FFT and PTRANS would present the highest interference levels.

However, specifically for PTRANS, our model points out that interference experienced by this application would vary significantly depending on the instance being solved. Our model indicated that PTRANS.I1.P6 would experience an interference level around 40% when co-located with itself in a two-by-two fashion, while PTRANS.I2.P6, when co-located in same conditions, would present an interference level of approximately 12%.

In addition, our model predicted that PTRANS and DGEMM, although belonging to the same Dwarf class, namely Dense Linear Algebra, would present distinct interference levels when co-located with themselves. So, prediction results indicated that the interference level resulted from co-location "PTRANS.I1.P6xPTRANS.I1.P6" would be approximately equal to 40%, while "DGEMM.I1.P6xDGEMM.I1.P6" would experience an interference level close to 3%.

Furthermore, our model predicted that FFT.I1.P4 and MUFITS.I1.P4, although presenting similar SLLC access rates, would experience distinct interference levels when co-located with themselves in a three-by-three fashion. Specifically, our solution predicted that FFT.I1.P4 would present a mutual interference around 40%, while MUFITS.I1.P4 would suffer a cross-interference approximately equal to 12%.

At last, although all co-locations used exactly the same number of virtual machines (twelve), the model predicted that those applications would experience distinct levels of interference.

3.4.3 Analysis of the Interference Prediction Model

In order to evaluate the quality of our proposed model, we executed all co-locations defined in schemes A, B, C and D in the physical server earlier described in Section 3.2.2, and, for each co-location, we calculated the *prediction error* achieved by our model. The prediction error is defined as the absolute value of the difference between the interference level predicted by our model and the real interference level obtained in a real scenario. The complete set of results achieved in these interference experiments are described in Appendix .

As can be seen in Figure 3.7, our model presented a median prediction error equal to 3.91%, and, in approximately 92% of all tested cases,the error was less than 10%. Remark that, in only 0.75% of all co-locations (5 occurrences), the prediction error was higher than

15%. Such results showed that our model was able to predict, for several co-locations, the interference level with a reasonable prediction error. Thus, the model was capable of estimating interference level for a set of real HPC applications with heterogeneous access profiles and computation patterns.

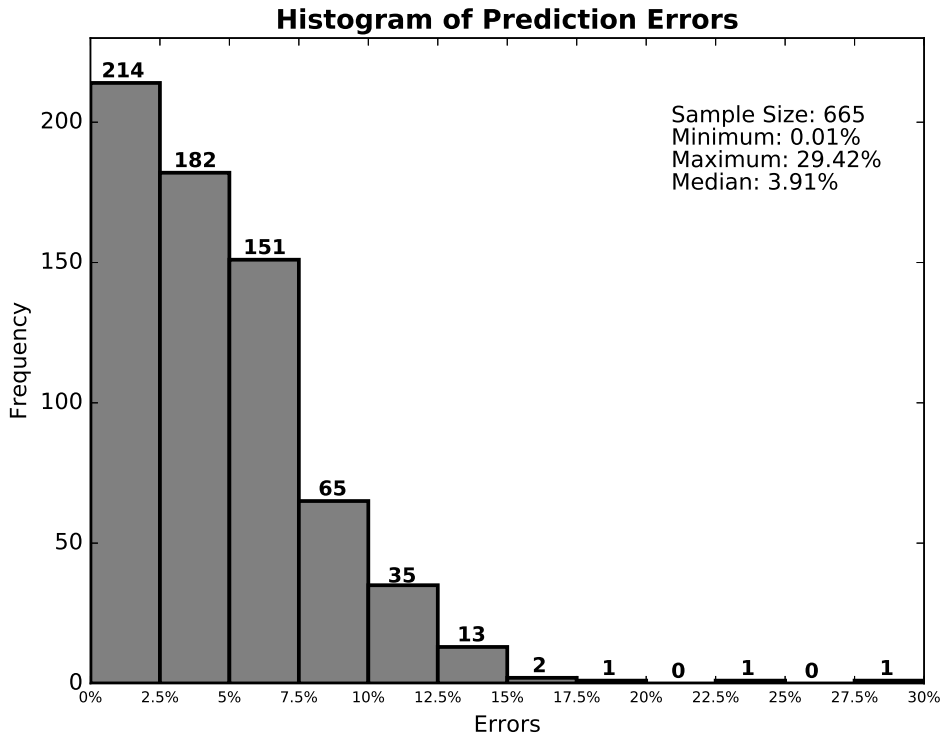


Figure 3.7: Frequency of prediction errors

Some interesting results are highlighted in Table 3.10. At first, as predicted by our model, experimental results showed that PTRANS, when executing with six virtual machines, really presented distinct interference levels when treating I1 and I2. Our model was able to predict interference of PTRANS regardless of the instance being solved because it takes into account the amount of accesses of applications to shared resources. Indeed, as can be seen in Table 3.9, the amount of individual access of PTRANS to all shared resources, when solving I1, is significantly higher than the one achieved when treating I2.

Besides that, those experimental results confirmed that PTRANS and DGEMM, even belonging to the same Dwarf class, presented distinct interference levels. This result allows to state that a qualitative approach based on Dwarfs classes is not enough to precisely determine interference. On the other hand, our model was able to predict that PTRANS and DGEMM would present, respectively, high and low interference levels.

Moreover, results also showed that SLLC access contention is not the only cause of

Co-location	Interference Level		Prediction Error
	Real	Predicted	
PTRANS.I1.P6xPTRANS.I1.P6	44.50%	39.97%	4.53%
PTRANS.I2.P6xPTRANS.I2.P6	5.31%	12.11%	6.80%
PKTM.I2.P6xPKTM.I2.P6	0.03%	0.20%	0.17%
DGEMM.I1.P6xDGEMM.I1.P6	7.79%	2.50%	5.29%
FFT.I1.P4xFFT.I1.P4xFFT.I1.P4	49.31%	40.45%	8.87%
MUFITS.I1.P4xMUFITS.I1.P4xMUFITS.I1.P4	22.85%	11.65%	11.20%

Table 3.10: Prediction errors for a subset of interference experiments with real applications

the cross-interference problem. FFT.I1.P4 and MUFITS.I1.P4, although having a similar amount of SLLC individual accesses, experienced distinct interference levels. As shown in Table 3.10, our model predicted satisfactorily the interference experienced by these applications. Although both applications present similar SLLC access rates, FFT.I1.P4, that experienced a higher interference, puts a higher pressure on virtual network than MUFITS.I1.P4.

Furthermore, our model correctly predicted that applications could experience distinct interference levels even when using the same number of virtual machines. Thereby, the number of virtual machines running in a physical host cannot be used as a parameter to determine the level of interference experienced by co-located applications.

All these findings allow to assert that solutions based only on (i) number of running virtual machines, (ii) Dwarfs classes or (iii) SLLC access contention are not suitable for determining interference. Our proposed model satisfactorily predicted interference for these specific cases because it takes into account the amount of simultaneous access to SLLC, DRAM and virtual network, besides considering similarities between applications access profiles.

3.5 Extending the Interference Prediction Model

Although the prediction model has achieved satisfactory results, we observed that the error increases with the rising of co-located applications. As shown in Figure 3.8, the median prediction error calculated for co-locations of scheme A was equal to 3.10%, whereas in scheme D the error raised to 7.95%. Besides that, as presented in Figure 3.9, prediction errors higher than 15% occurred exclusively in schemes C and D, that is, for cases where six or twelve applications were allocated to the same physical machine. Our model presented high prediction errors for these cases because it was built upon a dataset com-

prised only of results obtained from two-by-two fashion experiments. In other words, we supposed that only the accumulated access to shared resources and similarities among applications profiles were enough to predict interference, regardless of the number of co-located applications.

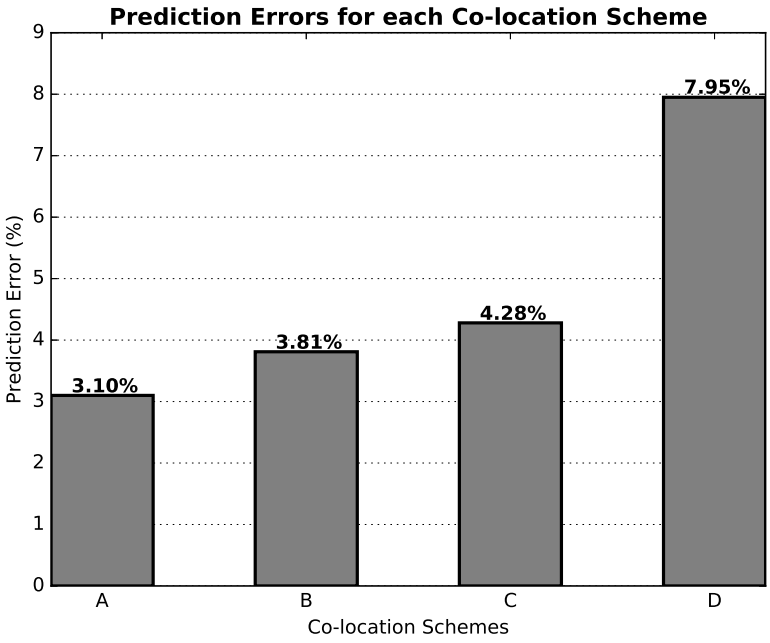


Figure 3.8: Median prediction errors for each co-location scheme

However, as can be seen in Table 3.11, the interference level is also affected by the number of applications allocated to the same physical machine. Consider, for example, applications MUFITS.I1.P6 and MUFITS.I1.P2 when co-located with themselves in a two-by-two and six-by-six fashion, respectively. Although co-location "6 x MUFITS.I1.P2" has a smaller amount of accumulated access to shared resources than "2 x MUFITS.I1.P6", the former presented a higher interference level than the latter. This possibly happens because the level of concurrency in co-location "6 x MUFITS.I1.P2" is higher than in "2 x MUFITS.I1.P6", that is, in the first co-location there are four more applications disputing over shared resources than in the second one.

		Accumulated Score			
Co-location	**Number App.**	**SLLC**	**DRAM**	**NET**	**Interference Level**
2 x MUFITS.I1.P6	2	0.11	0.25	0.02	10.72%
6 x MUFITS.I1.P2	6	0.10	0.16	0.00	22.95%

Table 3.11: Results of accumulated score and interference levels for a subset of experiments

Therefore, besides the amount of accumulated access to shared resources and similarity

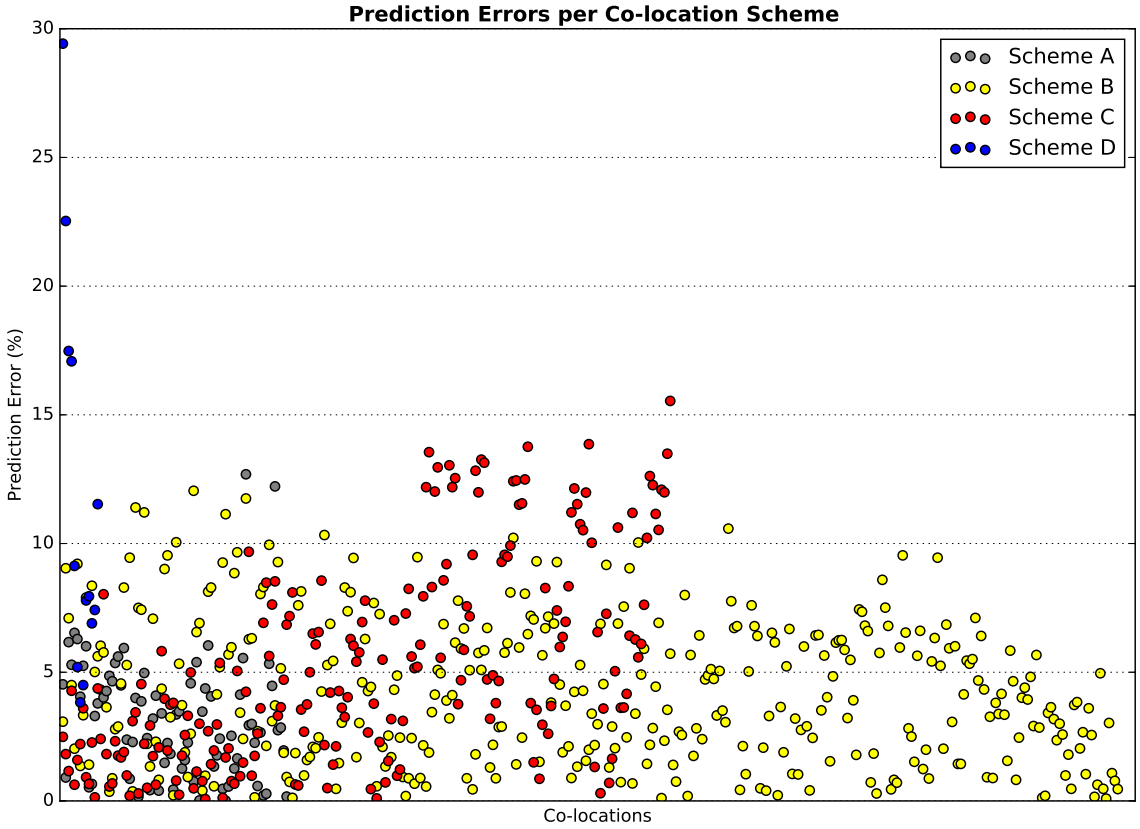


Figure 3.9: Prediction errors per co-location scheme

among applications profiles, this level of dispute over shared resources might also be considered when predicting interference for a set of co-located applications. In order to measure this level of concurrency, we defined the *concurrency factor* (R) in a physical machine m as the ratio between the number of applications allocated to m ($|A_m|$) minus 1 and the maximum number of applications that can be allocated to the physical machine (n) minus 1:

$$R_m = \frac{|A_m| - 1}{n - 1} \tag{3.9}$$

As this factor is used to measure the level of dispute over shared resources, it is calculated only for cases where more than one application are allocated to the same physical machine. Anyway, when just one single application is running in the physical machine, the prediction model is neither used since interference level, for those cases, is assumed to be equal to zero, as already stated in Section 3.1.

In order to verify the influence of the level of concurrency in the prediction model, we generated an extended interference dataset comprised of distinct values for the concurrency factor. This dataset was created by varying the number of synthetic applications

Application	Number of Virtual Machines	Individual Score		
		SLLC	DRAM	NET
S1	4	1.008	0.000	0.113
S3	4	0.119	0.009	0.109
S4	4	0.257	0.867	0.090
S7	4	0.922	0.036	0.495
S10	4	0.262	0.892	0.487
S12	4	0.713	0.059	0.494
S13	4	0.954	0.000	0.337
S16	4	0.257	0.867	0.326
S18	4	0.717	0.072	0.352
S3	2	0.066	0.000	0.026
S7	2	0.571	0.000	0.279
S10	2	0.502	0.070	0.309
S18	2	0.482	0.000	0.177
S3	1	0.033	0.000	0.000
S7	1	0.282	0.000	0.000

Table 3.12: SLLC, DRAM and virtual network individual scores of synthetic applications executed with a lower number of virtual machines

allocated to the same physical machine. In order to vary the number of co-located applications, we configured some applications from synthetic workload to be executed with a lower number of virtual machines, as listed in Table 3.12. By using applications with 6, 4, 2 and 1 virtual machines, we could allocate 2, 3, 6, and 12 applications to the same physical machine to accomplish several co-locating experiments along the lines of schemes A, B, C and D. So, we created the extended interference dataset by calculating, for each co-location, (i) the resulting interference level, (ii) the concurrency factor, (iii) the global similarity factors and (iv) the accumulated amount of access to shared resources.

From this new dataset and considering concurrency factor as an additional independent variable, we applied the MRA building process to create the extended interference prediction model:

$$\begin{aligned} F_m = &(0.5680 * T_{SLLC,m} * R_m) + \\ &(0.6758 * T_{SLLC,m} * G_{SLLC,m}) + \\ &(0.1422 * T_{NET,m} * G_{NET,m}) + \\ &(0.0516 * T^2_{DRAM,m} * G_{DRAM,m}) \end{aligned} \tag{3.10}$$

This parsimonious and statistically significant model presented a R^2-adj around 94.55% which is higher than the one achieved by the previous model (91.21%). As SLLC access contention is one of the major causes of interference, the concurrency factor was used in this new model to weight the amount of accumulated access to this shared resource ($T_{SLLC,m} * R_m$).

By using this extended model, we repeated the validation experiments with the real applications. We used the extended model for predicting the interference levels according to the co-location schemes A, B, C and D, and, for each co-location, we calculated the corresponding prediction error. The extended model achieved a median prediction error equal to 3.59% and, in approximately 93% of all tested cases, the error was less than 10%.

Moreover, the maximum error achieved by the extended model was equal to 13.88% which is significantly lower than the one presented by the previous model, around 29.42%. Moreover, as can be seen in Figure 3.10, the median prediction error remains almost the same for all co-location schemes. In short, those results indicate that, unlike the previous model, the extended model was able to deal properly with the negative effect that the number of co-located applications introduces in the interference level.

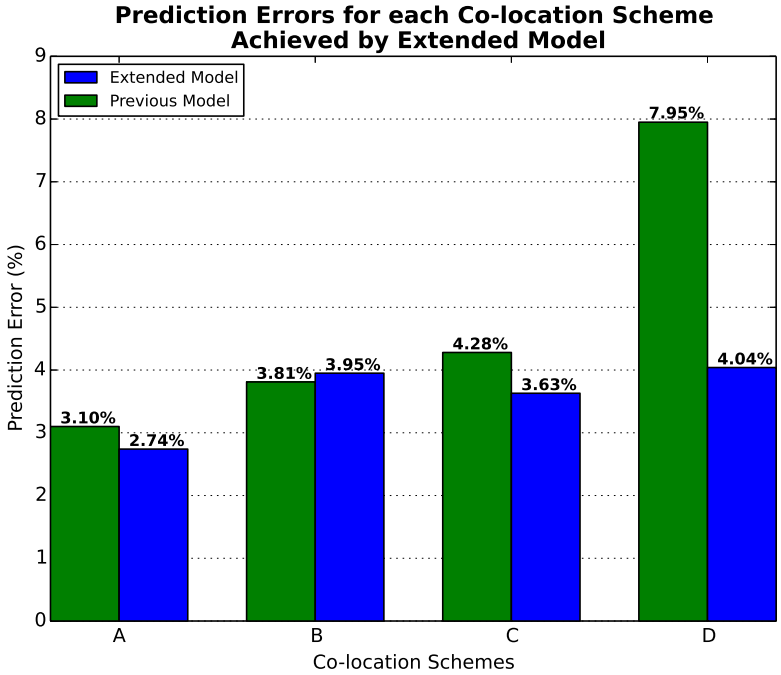


Figure 3.10: Median prediction errors for each co-location scheme achieved by the extended model

3.6 Comparing Extended and Gold Standard Prediction Models

The previous results showed that the extended model was able to predict the interference level experienced by a set of real HPC applications. Despite those satisfactory results, we devised an additional test to evaluate the precision of the extended model by comparing it with a gold standard model. This model was generated exclusively from the dataset provided by real experiments. Because it was built from the validation dataset, we suppose that this gold standard model provides the best predictions for this sample. From this test, we aim to evaluate how close the extended model is from the gold standard model in terms of prediction error.

Thus, we applied the MRA building process in the dataset provided by the real experiments in order to create the gold standard model:

$$\begin{aligned} F_m = &(0.2303 * T_{SLLC,m} * R_m) + \\ &(0.9048 * T_{SLLC,m} * G_{SLLC,m}) + \\ &(0.0652 * T_{NET,m} * G_{NET,m}) + \\ &(0.6804 * T^2_{DRAM,m} * G_{DRAM,m}) \end{aligned} \tag{3.11}$$

This model presented the same terms of the extended model and achieved an R^2-*adj* around 94.13%, practically the same of the extended model (94.55%).

As can be seen in Figure 3.11, the gold standard model achieved a median prediction error equal to 3.18% which is smaller than the one presented by the extended model. This is expected since the gold standard model was built and validated by using the same dataset, whereas the extended model was validated from an unseen dataset. Moreover, the interquartile range[5], calculated for both models, revealed that the results obtained from the gold standard model presented a smaller data dispersion than the extended model. This means that values are concentrated around the median prediction error which confirms the reliability of this value to represent the central tendency of data.

Although the gold standard model presented a lower median prediction error than the extended model, the difference between the median prediction errors for both models is not significant, i.e., less than 0.5%. Likewise, the small difference between the interquartile

[5]The interquartile range is a measure of statistical dispersion calculated from the difference between the first and third quartiles.

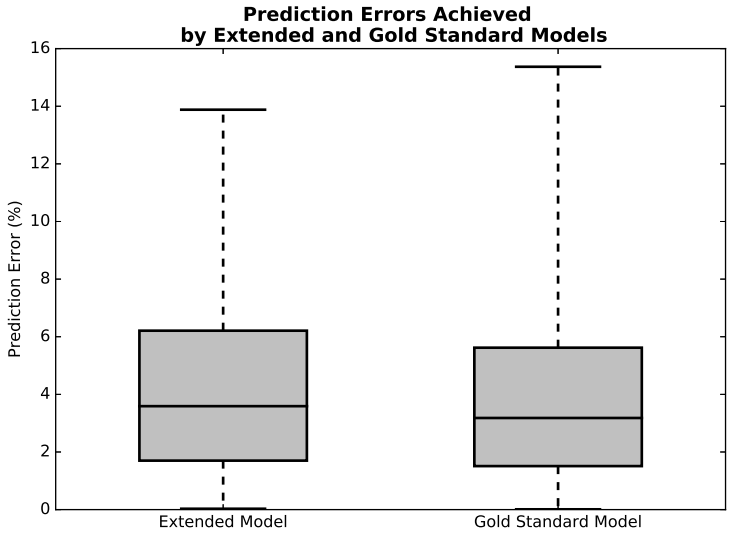


Figure 3.11: Prediction errors achieved by extended and gold standard models

range, around 0.4%, points out that the variations in both samples are similar.

Those results confirmed that the extended model really achieved very good predictions for the validation dataset, even being built upon an interference dataset generated from synthetic applications. In short, this analysis indicates that the model is capable of performing satisfactory out-of-sample predictions.

Chapter 4

Efficient Usage of Physical Machines

IVMPP has two objectives: (i) minimizing the interference experienced by applications that share a same physical machine and (ii) minimizing the total number of physical hosts required to allocate those applications in the cloud environment. The first goal aims to rise the performance of HPC applications when executed in clouds, while the second one, besides reducing energy costs, allows cloud provider to rapidly deliver computational resources.

In this chapter, we present the criterion adopted by our proposal to achieve that second objective, i.e., to accomplish an efficient usage of physical resources in such a way that the number of active physical machines can be minimized. First, we briefly discuss in Section 4.1 the relevance of minimizing the number of used physical machines in clouds. Next, in Section 4.2, we show that the traditional approach, commonly used to achieve this goal, may not be suitable for dealing properly with the VMP problem. In Section 4.3, we finally describe the criterion used by our VMP strategy to minimize the number of used physical machines through a balanced and efficient usage of resources.

4.1 Background

A typical cloud computing environment is comprised of a set of physical machines, where each of them is equipped with some virtualization mechanism. This virtualization mechanism provides virtual machines where users can execute their own operating system and applications [15]. A physical machine is said to be active, or in use, when it holds, at least, one single virtual machine. Otherwise, that physical machine is considered as being unused or inactive.

Minimizing the number of used physical machines is crucial in cloud environments for two main reasons. At first, the cloud provider can promptly satisfy incoming requests since there will be, possibly, a greater number of physical machines available to host new virtual machines. In other words, it can provide the illusion of infinite resource capacity. This capacity, though theoretical, allows customers to rapidly obtain computational resources without considering constraints related to physical space, energy provisioning and cooling capacity. This advantage eliminates the need for users to plan far ahead for provisioning computational resources required to deploy their applications in clouds [17]. Secondly, it also allows to reduce the total energy consumption in a datacenter. An unused physical machine can either be switched off or put in power saving mode, thus reducing the energy needed to keep a datacenter working in a given time interval [22]. As a consequence, cloud providers can decrease its TCO (Total Cost of Ownership) by dwindling energy operational costs. Moreover, as a high percentage of carbon emissions is directly attributed to energy consumption[1], the reduction of power consumption can also contribute to slow down the progress of global climate change [86].

In order to achieve a minimal usage of physical machines, a VMP strategy must accomplish an efficient and balanced usage of physical resources. Consider, for example, two scenarios illustrated in Figure 4.1. In both scenarios, namely A and B, the cloud computing environment is comprised of three physical machines, each one with 10 CPUs. In scenario A, after six allocation requests (time *t6*), the physical machines M1 and M2 can only host virtual machines with 1 and 2 CPUs, respectively. Thus, to host the virtual machine with 3 CPUS, requested in *t7*, the VMP strategy should necessarily use an addition physical machine, M3. But, if the allocation of virtual machines was accomplished as illustrated in scenario B, that virtual machine with 3 CPUs could be promptly allocated in M1, thus avoiding to activate a new physical machine to satisfy that incoming request.

Therefore, as shown by that simple example, the virtual machine placement strategy can significantly influence the number of physical machines needed to serve all allocation requests.

[1]About 98 % of CO^2 emissions (or 87 % of all CO^2-equivalent emissions from all greenhouse gases) can be directly attributed to energy consumption, according to a report by the EIA (Energy Information Administration) [86].

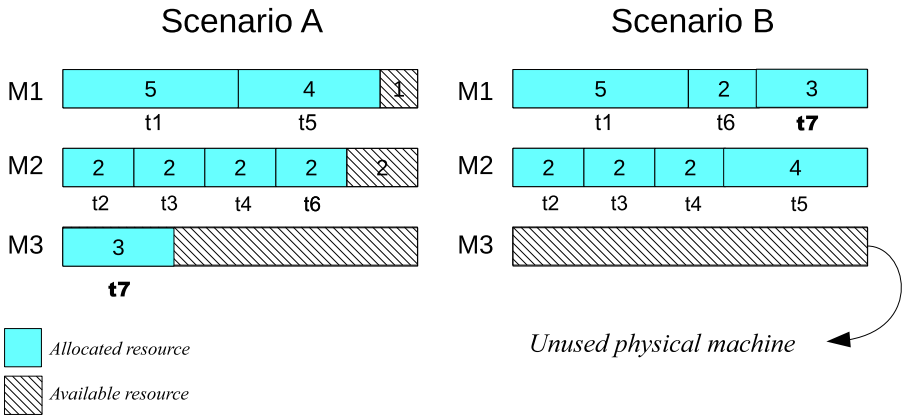


Figure 4.1: Relation between efficient usage of resources and minimization of used physical machines

4.2 Vector Bin Packing

Several works, such as [55] and [61], proposed solutions based on the classical Bin Packing (BP) problem to minimize the number of physical machines in clouds. In this problem, a set of items of various sizes has to be packed into the smallest number of bins [39] [63]. In the context of VMP problem, items and bins are used to represent, respectively, virtual and physical machines. Moreover, dimensions of items and bins correspond, in the VMP problem, to physical resources requested and delivered by virtual and physical machines, respectively. These physical resources can be, for example, the number of CPUs and amount of main memory.

When considering just one dimension (physical resource, in case of VMP problem), both VMP and BP problems are very similar to each other. However, VMP cannot be tackled as a BP problem for cases where more than one dimension is being evaluated. Notice that, though two-dimension BP allows to place items side-by-side or one above the other, this is not a valid operation in the context of the VMP problem. Such operation cannot be executed in VMP context because a virtual machine cannot use the same physical resource already assigned to other virtual machine. In other words, once the physical resource is allocated to a virtual machine, it cannot be reallocated to another one [67] [75]

To clarify this issue, consider the example illustrated in Figure 4.2. In this example, the Cartesian plane is used to represent the allocation of virtual machines to a physical machine, where abscissa and ordinate axes correspond to the number of CPUs and amount of main memory, respectively. Note that, in such example, new virtual machines can only

be placed in the area indicated by the hatched rectangle. Indeed, this is the unique area where both resources are not allocated to any virtual machine. Conversely, if a new virtual machine was placed in areas marked with "X", there would be a resource overlapping, i.e., two virtual machines would be using the same physical resource.

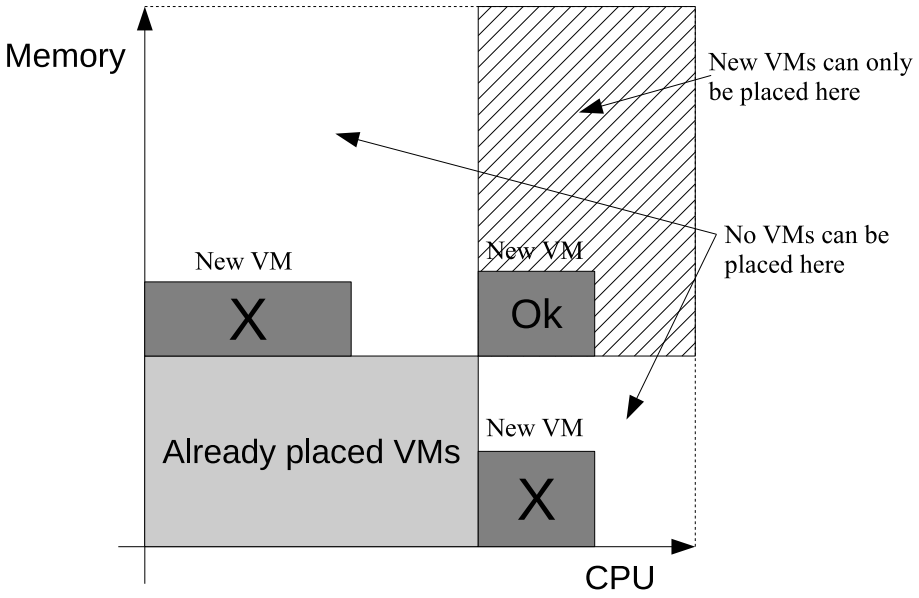


Figure 4.2: VMP in two-dimension resource space [67]

Before this anomaly, some works postulated that an approach based on a vector of resources could be more suitable for dealing with the VMP problem [43] [67] [75]. In this novel approach, called Vector Bin Packing (VBP), the available physical resources are represented as multidimensional normalized vectors, where each dimension corresponds to a physical resource. Similarly, requested resources for virtual machines are also represented in the same way. By treating the amount of requested and available resources as a vector, the aforementioned problem of overlapping physical resources is avoided since each physical resource is individually compared and evaluated.

4.3 Allocation Metrics

In this work, we used the following vectors of resources to address the VMP problem: (i) remaining resources of the physical machine ($\overrightarrow{REM}$), (ii) requested resources ($\overrightarrow{REQ}$) and (iii) remaining resources after allocating requested resources ($\overrightarrow{RAF}$). Each of these vectors has two elements, corresponding to the amount of CPU and main memory. As the amount of CPU (number of cores) and main memory (gigabytes) assumes distinct range of values, we normalized them with respect to the corresponding total amount of

resources in the physical machine. For example, a CPU and a memory requests equal to 0.5 means that the application will allocate half of the total amount of CPU and main memory provided in the physical machine.

From these vectors, we used the following two metrics to determine the best physical machine to allocate a set of applications. The first one, called *alignment factor*, was also used in [43] and is calculated as the angle between vectors $\overrightarrow{REQ}$ and $\overrightarrow{REM}$:

$$\theta = acos\left(\frac{(REQ.cpu * REM.cpu) + (REQ.mem * REM.mem)}{(\sqrt{REQ.cpu^2 + REQ.mem^2}) * (\sqrt{REM.cpu^2 + REM.mem^2})}\right) \quad (4.1)$$

This factor is used to evaluate the shape of the exploitable volume of resources, allowing to assess whether required and remaining resources are complementary to each other. To provide a better understanding on this metric, consider, for example, the scenario presented in Figure 4.3. In such example, a VMP strategy should select, from two physical machines - M1 and M2, the best one to allocate a virtual machine requesting 0.3 and 0.2 of CPU and main memory, respectively. Suppose that the amount of physical resources, left on each physical machine, is indicated by the values of vector $\overrightarrow{REM}$. Since both physical machines have enough resources to allocate the new virtual machine, the decision comes down to choosing the one that will yield the smaller resource wastage.

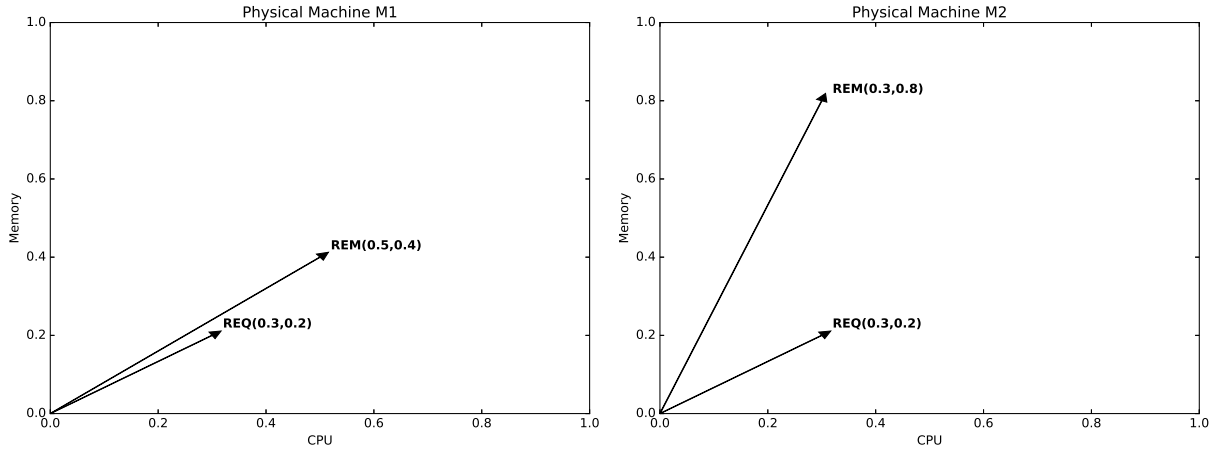


Figure 4.3: Example of resource allocation

By calculating the alignment factor for both physical machines, we can assert that M1 would be the best choice for that case. As can be seen in Figure 4.3, vectors $\overrightarrow{REM}$ and $\overrightarrow{REQ}$, in M1, present a smaller angle if compared with the same vectors in M2. This means that the amount of resources required by the new virtual machine fits better to the amount of resources left in M1. In fact, the incoming virtual machine requires a greater

amount of CPU than main memory and, in a complementary way, the physical machine M1 presents a greater amount of remaining CPU than main memory.

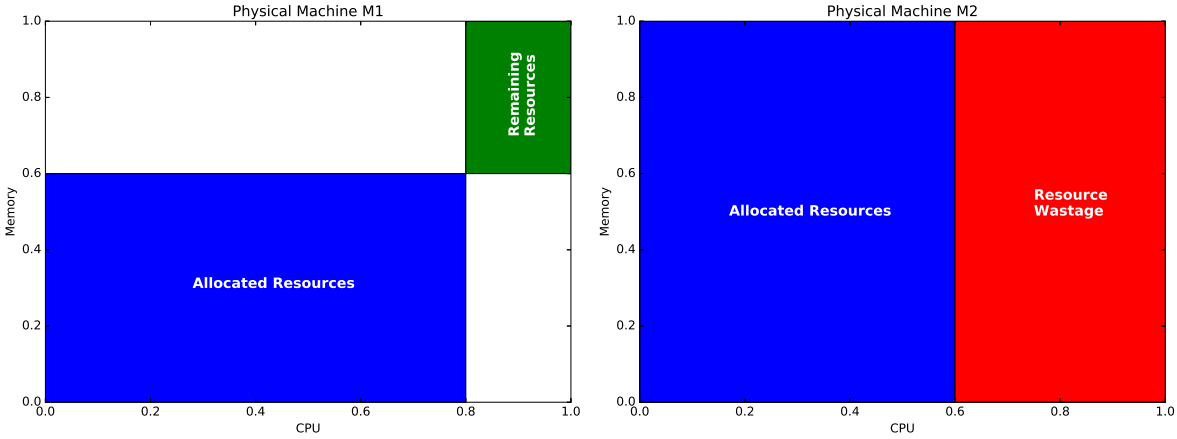


Figure 4.4: Resultant resource usage after allocation supposed in Figure 4.3

Note that, if the new virtual machine was placed in M2 instead of M1, there would be a resource wastage in that physical machine. As can be seen in Figure 4.4, this resource wastage comes from the fact that 40% of CPU available in that physical machine cannot be longer used for any virtual machine. On the other hand, in case of M1, we can observe a balanced use of resources since there would be almost the same amount of remaining main memory and CPU.

The other metric, called *residual factor*, is equal to the length of vector $\overrightarrow{RAF}$:

$$\tau = \sqrt{RAF.cpu^2 + RAF.mem^2} \tag{4.2}$$

This metric is used to assess the size of exploitable volume of resources, i.e, it measures the amount of resources left in the physical machine after the allocation of the requested resources. This second metric is used as a tiebreaker for the first metric, i.e., for cases with the same alignment factor, the residual factor is used to select the physical machine which will result in the smallest resource wastage.

For example, consider the scenario illustrated in Figure 4.5, where two physical machines, namely M1 and M2, have allocated the respective amount of resources indicated by vectors $\overrightarrow{REM}$. In addition, suppose that a new virtual machine, requesting 0.3 of CPU and main memory, should be placed in one of those physical machines. Note that, though the alignment factor is exactly the same for both physical machines (0.00), the allocation of requested resources in M2 would result in a more efficient usage of physical resources. In fact, as shown in Figure 4.6, resources in M2 would be fully allocated, whereas M1

would still have a small amount of remaining resources. Depending on the minimal size of virtual machines offered by the cloud, this small piece of remaining resources may not be even allocated, thus resulting in resource wastage.

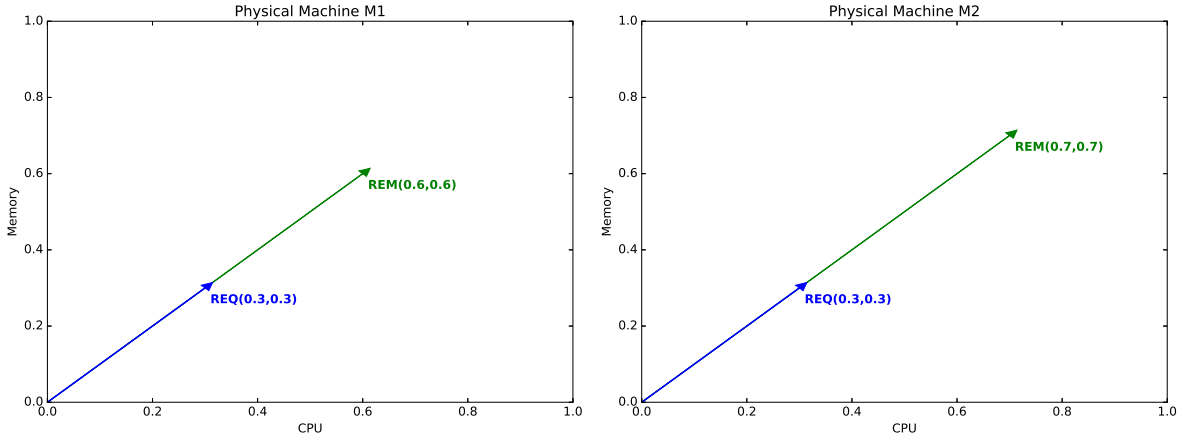


Figure 4.5: Example of resource allocation in two physical machines.

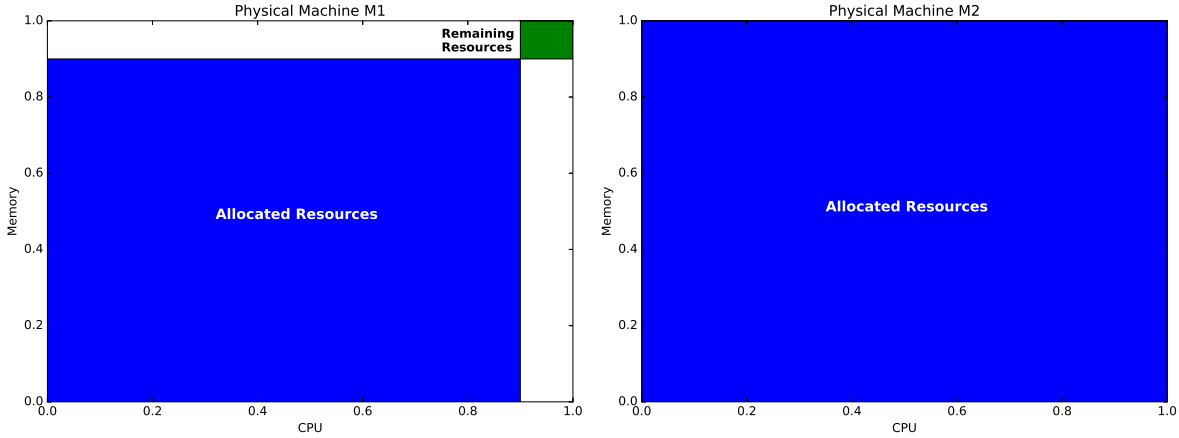


Figure 4.6: Resultant resource usage after allocation supposed in Figure 4.5

In short, when allocating a new set of virtual machines, our VMP strategy first seeks for the physical machine which results in the smallest alignment factor. However, in case of a tie, the VMP strategy selects, among the physical machines with the same alignment factor, the one which presents the smallest residual factor. We claim that this criterion is suitable for accomplishing a balanced and efficient usage of physical resources because it considers both the shape (alignment factor) and the size (residual factor) of the exploitable volume of resources available in the cloud environment.

Chapter 5

An Interference-aware Virtual Machine Placement Strategy

In this chapter, we formally introduce the IVMPP in Section 5.1, while in Section 5.2 we describe our proposed strategy to solve it. At last, in Section 5.3, we describe experimental tests accomplished to evaluate our proposal.

5.1 Mathematical Formulation of IVMPP

In this section, we present a mathematical formulation for the IVMPP whose notations are summarized in Table 5.1. Let M be the set of physical machines available in a cloud environment and A the set of applications to be allocated in that environment. Moreover, we define the total amount of main memory and number of CPUs provided by each physical machine as Mem and Cpu, respectively. Similarly, the amount of CPU and main memory required by an application $i \in A$ is defined as m_i and c_i, respectively. It is worth mentioning that, in this work, we assume that all physical machines have the same hardware configuration, i.e., they are equipped with the same amount of CPU and main memory.

Moreover, we define γ_Q as the interference level experienced by a subset of applications $Q \subseteq A$ when allocated to a same physical machine and ω as the maximum interference level that can be reached on each physical machine. We now define a binary variable $X_{i,j}$ for each $i \in A$ and $j \in M$ such that $X_{i,j}$ is equal to 1 if and only if application i is allocated to the physical machine j, and equal to 0, otherwise. In addition, we define a binary variable Y_j for each $j \in M$ such that Y_j is equal to 1 if and only if the physical machine j is being used, i.e., if there is, at least, one application allocated to it. Y_j is equal

to 0 if no application is allocated to j. This problem can be formulated as the integer programming problem described in the following equations:

$$min \left(\left\{ \frac{\sum_{j \in M} \left[\sum_{Q \subseteq A} \left(\prod_{i \in Q} X_{i,j} \right) . \gamma_Q \right]}{\omega . \sum_{j \in M} Y_j} \right\} . \alpha + \left(\frac{\sum_{j \in M} Y_j}{|M|} \right) . (1 - \alpha) \right) \tag{5.1}$$

$$\sum_{i \in A} X_{i,j} = 1, \forall i \in A \tag{5.2}$$

$$\sum_{i \in A} X_{i,j} . m_i \leq Mem . Y_j, \forall j \in M \tag{5.3}$$

$$\sum_{i \in A} X_{i,j} . c_i \leq Cpu . Y_j, \forall j \in M \tag{5.4}$$

$$X_{i,j} \leq Y_j, \forall i \in A, \forall j \in M \tag{5.5}$$

$$Y_{j+1} \leq Y_j, \forall j = \{1...|M| - 1\} \tag{5.6}$$

$$X_{i,j} \in \{0, 1\}, \forall i \in A, \forall j \in M \tag{5.7}$$

$$Y_j \in \{0, 1\}, \forall j \in M \tag{5.8}$$

The objective function defined in Expression 5.1 seeks for minimization of the sum of interference levels presented in each physical machine and the number of machines used to allocate all applications. Note that the sum of interference levels was normalized with respect to the maximum sum of interference levels that can be reached in our environment, i.e., $\omega . \sum_{j \in M} Y_j$.

Besides that, we normalized the number of used machines concerning the total number of physical machines available in the cloud environment. As both objectives were normalized between 0 and 1, we can use the weight α to determine the relevance of each objective. Therefore, when α is set to 1 the objective function prioritizes the minimizing

Notation	Description
M	Set of available physical machines
A	Set of applications to be allocated
Mem	Total amount of memory on each physical machine
Cpu	Total amount of CPU on each physical machine
m_i	Amount of memory requested by application i
c_i	Amount of CPU requested by application i
γ_Q	Interference level experienced by subset of applications Q
ω	Maximum interference level achieved in the environment
$X_{i,j}$	Variable is equal to 1 whether application i is allocated to machine j
Y_j	Variable is equal to 1 whether physical machine j is being used
α	Weight to determine the relevance of each objective

Table 5.1: Description of notations used in mathematical formulation for the IVMPP

of the sum of interference levels, and when α is close to 0 the function will minimize the number of used physical machines. Thus, this parameter can be adjusted to reach a desirable trade-off between both objectives.

Constraints described in Equation 5.2 ensure that each application is entirely allocated to just one physical machine. Inequalities defined in (5.3) and (5.4) enforce that the total amount of CPU and main memory available in physical machines are not exceeded. In Inequalities (5.5) we defined constraints to guarantee that a physical machine is used if and only if it holds, at least, one single application. As all physical machines permutation yields feasible solutions, we added a set of constraints, defined in (5.6), that eliminates symmetry by establishing an order at which machines shall be used in the cloud environment. At last, constraints described in (5.7) and (5.8) define the binary and integrality requirements on the variables.

5.2 Multistart Iterated Local Search for the IVMPP

The classic VMP problem is classified as a NP-Hard problem [74]. So, IVMPP, that is a variant of VMP, can also be classified as a NP-Hard problem. For this sort of problem, exact procedures have often proven to be incapable of finding solutions in a reasonable time.

Therefore, we propose in this work a solution based on the Iterated Local Search (ILS) framework that belongs to the class of local search algorithms such as VNS (Variable Neighborhood Search) and GRASP (Greedy Randomized Adaptive Search Procedure) [58]. As depicted in Figure 5.1, a metaheuristic based on ILS firstly creates an initial

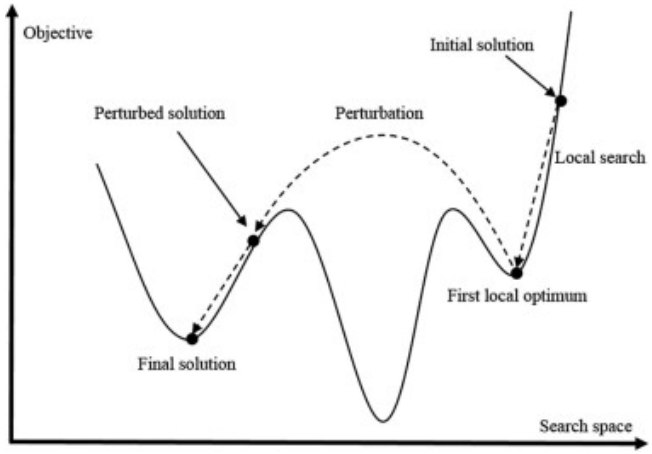


Figure 5.1: Iterated Local Search

solution to be submitted to a local search procedure. This procedure aims to find the best solution in a given neighborhood. So, to escape from local optima, the ILS applies perturbations on the current solution in order to provide the diversification needed to find the global optimum solution. Moreover, in this work, we adopted a multistart approach where new initial solutions are submitted to ILS, allowing it to explore larger areas of the search space.

The input data for our method is comprised of (i) the set of applications to be allocated to the cloud (A), containing for each of them the amount of requested CPU and memory, and the individual access to each of the three shared resources, and (ii) the set of physical machines (M). Notice that the amount of physical resources, CPU and main memory, were normalized with respect to the total amount of resource provided by the physical machine.

In order to present our proposed algorithm, we need to introduce some additional notation. We define a placement solution $s = \{(a1, m1), (a2, m1), (a3, m2)...\}$ as the set of 2-tuples (a, m) representing that the application a is allocated to physical machine m. Moreover, we define $Z(s)$ as the normalized sum of interference levels predicted for solution s. This sum of interference levels is calculated by estimating the level of interference presented by each physical machine of solution s. To estimate this level of interference, we used the extended prediction model earlier described in Section 3.5.

In addition, we define $M(s)$ as the normalized number of physical machines used in s, and $E_{CPU}(s)$ and $E_{MEM}(s)$ as the percentage of amount of CPU and main memory exceeded in solution s, respectively. A solution is considered feasible if and only if $E_{CPU}(s)$ and $E_{MEM}(s)$ are equal to zero.

We define a cost function $f : \mathbb{S} \rightarrow \mathbb{R}$, where $\mathbb{S}$ is the set of all possible solutions, as follows:

$$f(s) = Z(s).\alpha + M(s).(1 - \alpha) + E_{CPU}(s).\lambda + E_{MEM}(s).\lambda \tag{5.9}$$

Similarly to the objective function described in the math model, this function $f(s)$ attempts to minimize the sum of levels of interference and the number of used machines. In addition, $f(s)$ penalizes, in accordance with parameter λ, unfeasible solutions, which use more CPU and memory than the available amount in the environment.

Our proposed solution, called IVMPP_ILS, is described in Algorithm 2. For each iteration of the *Multistart Loop* (lines 2 to 15), the algorithm executes the procedure *ConstructivePhase* (line 3) to create a new solution to be submitted to the ILS. Then, *ILS Loop* (lines 4 to 13) performs a local search in the neighborhood of the current solution through *Variable Neighborhood Descent* (VND) method (line 5). Both VND and *ConstructivePhase* are described in detail later.

After executing VND, IVMPP_ILS evaluates the quality of the current solution $\overline{s}$ (line 6). If $\overline{s}$ represents an improvement on the best current solution s^*, it is set as the actual best solution and the counter j is reset to 1 (lines 7 and 8). Otherwise, the counter j is incremented (line 10). Then, the algorithm executes the procedure *Perturbation* (line 12) that applies a perturbation on the current solution in order to escape from local optima.

Perturbation executes j random movements of three classical local search movements for the original VMP problem: *Move1*, *Move2* and *Swap1*. Heuristics *Move1* and *Move2*, move, respectively, one and two applications to another used physical machine, while *Swap1* performs one swap operation, i.e., it swaps two applications between two distinct physical machines. Furthermore, this procedure also adds a new physical machine to the solution, before executing the last perturbation (i.e., when j is equal to *iterMaxILS*) of a given initial solution (i.e., solution created by *ConstructivePhase*).

Algorithm 3 presents the *ConstructivePhase*. Firstly, the algorithm sorts the list of applications in decreasing order (line 1) by considering the following sequence: (i) amount of accesses to SLLC, (ii) amount of requested CPU and main memory, and (iii) amount of accesses to virtual network and DRAM. Next, the algorithm inserts one physical machine into the set of used machines (lines 2 and 3). Then, the procedure executes *Allocation Loop* (lines 4 to 20) that basically performs two steps: (i) the selection of two applications from the list of sorted applications and (ii) the selection of a physical machine to allocate

Algorithm 2 IVMPP_ILS

```
    Input: A,M,β,iterMaxILS, iterMaxMultiStart
    Output: s*
 1: s* = ∅; f(s*) = ∞; i = 1; j = 1;
    /*Multistart Loop*/
 2: while i ≤ iterMaxMultiStart do
 3:     s = ConstructivePhase(A, M, β);
        /*ILS Loop*/
 4:     repeat
 5:         s̄ = VND(s, A, M);
 6:         if (f(s̄) < f(s*)) then
 7:             s* = s̄;
 8:             j = 1;
 9:         else
10:             j = j + 1;
11:         end if
12:         s = Perturbation(s̄, j);
13:     until j ≤ iterMaxILS
14:     i = i + 1;
15: end while
16: return s*
```

them.

The process of selecting this pair of applications is described as follows. Firstly, the algorithm creates the sets of applications $First$ and $Last$ that are composed, respectively, of the first and last N_a elements of the sorted list of applications (lines 5 to 7). This number of applications is defined by the parameter β that is used to control the degree of randomness of this constructive phase. Next, the algorithm selects, randomly, applications i and j from $First$ and $Last$, respectively (line 8). By selecting applications from the head and tail of the sorted list, this process aims to co-locate applications with complementary access profiles and amount of requested resources, thus resulting in a low interference level.

After selecting this pair of applications, the best physical machine to allocate them is chosen among the set of physical machines with enough resources to allocate both applications (line 10). The method selects the best physical machine by using the criterion described in Section 4.3, i.e., the physical machine which would present the smallest alignment and residual factors, in this order.

However, if none of the physical machines, already in use, has available resources to allocate applications i and j, the *ConstructivePhase* takes a new physical machine from M to allocate them (lines 12 to 14), i.e., the method activates a new physical machine to

Algorithm 3 *ConstructivePhase*

Input: A,M,β
Output: s
1: $App = Sort(A)$;
2: $Used = Used \cup \{m_1\}$;
3: $M = M \backslash \{m_1\}$;
/*Allocation Loop*/
4: **while** $App \neq \emptyset$ **do**
5: $N_a = \lceil \beta . |App| \rceil$;
6: $First =$ first N_a elements of App;
7: $Last =$ last N_a elements of App;
8: *Choose randomly applications* $i \in First$ *and* $j \in Last$;
9: **if** There is $m \in Used$ with enough resources to allocate i and j **then**
10: *Choose the best one to allocate* i *and* j;
11: **else if** $M \neq \emptyset$ **then**
12: *Pick up any* $m \in M$
13: $Used = Used \cup \{m\}$;
14: $M = M \backslash \{m\}$;
15: **else**
16: *Choose randomly* $m \in Used$;
17: **end if**
18: $s = s \cup \{(i, m), (j, m)\}$;
19: $App = App \backslash \{i, j\}$;
20: **end while**
21: **return** s

allocate both applications. This approach aims to reduce the number of used machines because it takes new machines only when it is absolutely necessary. Moreover, if none of physical machines is capable of allocating that pair of applications, the method chooses randomly a physical machine to hold them (line 16). At last, both applications i and j are allocated to the selected physical machine and removed from the sorted list of applications (lines 18 and 19). Notice that unfeasible solutions can be generated by this constructive method. However, their corresponding costs are very high due to the penalty applied in the proposed cost function.

At last, Algorithm 4 presents the VND method. This method executes iteratively these classical local search movements for the original VMP problem: the *Move1*, *Swap1*, and *Move2*, in this order. As we adopted the strategy of first improving, these heuristics stop executing upon improving the current solution. In addition, the VND halts when no improvement is achieved in these neighborhoods.

Algorithm 4 *VND*

Input: s,A,M
Output: $\overline{s}$
1: $M1 = S1 = M2 =$ True;
2: **while** $M1$ or $S1$ or $M2$ **do**
3: $M1 = S1 = M2 =$ False;
*/*Starts executing Move1(), the simplest local search*/*
4: $s' = Move1(s, A, M)$;
5: **if** $(f(s') < f(s))$ **then**
6: $s = s'$;
7: $M1 =$ True;
8: **end if**
*/*If Move1() not improved current solution, executes Swap1()*/*
9: **if** (**not** $M1$) **then**
10: $s' = Swap1(s, A, M)$;
11: **if** $(f(s') < f(s))$ **then**
12: $s = s'$;
13: $S1 =$ True;
14: **end if**
15: **end if**
*/*If neither Move1() nor Swap1() succeeded, executes Move2()*/*
16: **if** (**not** Mov1 **and not** Swa1) **then**
17: $s' = Move2(s, A, M)$;
18: **if** $(f(s') < f(s))$ **then**
19: $s = s'$;
20: $M2 =$ True;
21: **end if**
22: **end if**
23: **end while**
24: $\overline{s} = s$;
25: **return** $\overline{s}$

5.3 Experimental Tests and Results

We carried out experiments to assess our proposal concerning its ability for reducing the sum of interference levels, while keeping a low number of used machines. Moreover, we executed additional experiments to verify the optimality of solutions of our proposal, by comparing them with the ones given by an exact procedure.

In Section 5.3.1, we present the set of instances used as input for the IVMPP in our analysis. The comparison between IVMPP_ILS and the most used heuristics to minimize the number of used machines is discussed in Section 5.3.2. At last, in Section 5.3.3, we evaluate our proposal with respect to its capability of finding near-optimal solutions.

5.3.1 Generating Instances for the IVMPP

In order to evaluate our proposal, we created a set of instances for the IVMPP by considering the real workload described in Section 3.4.1.

An instance for the IVMPP is composed of (i) the set of applications to be allocated in the cloud environment and (ii) the number of physical machines available in that environment. For each application, the amount of individual access to SLLC, DRAM and virtual network, and the normalized amount of requested CPU and main memory are also given as input to the problem. In this work, we considered that all physical machines available in the cloud environment has the same hardware configuration, i.e., they have the same amount of CPU and main memory. As the amount of requested resources are normalized with respect to the total amount of resources available in physical machines, we could abstract this latter information from the IVMPP instance. Thus, if one application requires 0.5 of CPU, it is requesting 50% of the total amount of CPU available in the physical machine. Notice that this approach prevents the VMP strategy from being concerned about the absolute number of cores or amount of main memory available in the physical machines.

		Individual Access			**Required Resources**	
App. Label	**Number of VM**	**SLLC**	**DRAM**	**NET**	**CPU**	**Memory**
PTRANS.I1.P6	6	0.183	0.214	0.322	0.50	0.50
FFT.I2.P4	4	0.070	0.164	0.517	0.33	0.33
DGEMM.I3.P1	1	0.003	0.005	0.000	0.08	0.08
HPL.I2.P6	6	0.028	0.057	0.019	0.50	0.50
MUFITS.I1.P2	2	0.016	0.027	0.000	0.17	0.17
Number of Available PM	**5**					

Table 5.2: Example of instance for the IVMPP

Consider, for example, the instance illustrated in Table 5.2. In this example, five applications should be allocated in a cloud environment whose number of available physical machines is also equal to five. Note that the amount of requested resources are normalized with respect to the total amount of physical resources available in the machine used in our experiments.

We created 100 instances for the IVMPP, where the number of applications at each instance varied from 5 to 50 (interval of 5). More specifically, we generated 10 instances

with 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 applications. Each instance is a sample of applications randomly selected from the workload described in Section 3.4.1. It is worth mentioning that an application can appear more than once in a same instance.

In addition, for all instances, the number of machines available in the cloud environment was set as the total number of applications to be allocated. For example, the number of physical machines of the instance illustrated in Table 5.2 is equal to 5 since this is exactly the same number of applications to be allocated in the cloud. From this approach, we can evaluate whether IVMPP_ILS sacrifices the minimization of physical machines in favor of the minimization of interference. Indeed, by adopting this configuration, IVMPP_ILS has the option to spread out the applications, using all physical machines, in order to fully avoid interference in the cloud environment.

5.3.2 Comparing IVMPP_ILS with Classical Heuristics

The following heuristics are widely used to minimize the number of machines when allocating virtual machines in clouds: Best Fit (BF), First Fit (FF), Worst Fit (WF), Best Fit Decreasing (BFD), First Fit Decreasing (FFD) and Worst Fit Decreasing (WFD). Since these greedy algorithms have shown to be effective for solving this problem in previous works [51] [19], they were also used as a baseline here, in our experiments.

This experimental evaluation was executed in two phases. At first, we performed several offline experiments to calibrate the value of parameter α, i.e., to find out the α that gives a good trade-off between reducing both interference and the number of used physical machines. After that, by using the calculated α, we executed some experiments in a real scenario to verify the accuracy of the results achieved in the offline tests.

5.3.2.1 Calibrating Parameter α

The parameter α is used to adjust the behavior of IVMPP_ILS so that it can minimize, as much as possible, both objectives. Thereby, initially, several offline experiments were executed to calibrate such parameter.

We tested our proposal and the other heuristics over the set of instances earlier described in Section 5.3.1. In order to assess our proposal against those heuristics, we devised the following two metrics. In the first one, we determined the percentage of test cases where our solution achieved a *strictly smaller sum of interference levels* than the one reached by the heuristic. Concerning the minimization of interference, this metric

allows to quantify the number of test cases where our solution outperformed the tested heuristic.

In the second metric, we calculated the percentage of cases where our proposal used a *smaller or equal number of physical machines* than the one used by the heuristic. By using this metric, we expect to evaluate the number of cases where our solution achieved the minimal number of physical machines needed to allocate all applications in the cloud environment.

Note that, when comparing with a single heuristic, the best trade-off between minimizing interference and the number of physical machines is achieved when both metrics, reach, simultaneously, 100% of evaluated instances. In this hypothetical scenario, IVMPP_ILS would reduce interference for all instances, while keeping, at least, the same number of used physical machines given by the heuristic.

Concerning IVMPP_ILS, the input parameter α, which allows to control the relevance of each objective, was varied from 0.00 to 1.00 (interval of 0.10). By varying this parameter, we can verify the behavior of IVMPP_ILS with respect to minimizing both interference and number of physical machines. The remaining input parameters of IVMPP_ILS were defined from an empirical evaluation. So, parameters β, λ, *iterMaxILS* and *iterMaxMultistart* were fixed in 0.4, 0.5, 10 and 50, respectively. As IVMPP_ILS is a non-deterministic algorithm, we repeated the execution of each instance until reaching a coefficient of variation smaller than 2%. Moreover, we restricted the execution time of IVMPP_ILS in all cases to 15 seconds.

All codes were compiled with GCC (Gnu C Compiler) version 4.9.2 with optimization flag *O3* which enables the process of automatic vectorization [62].

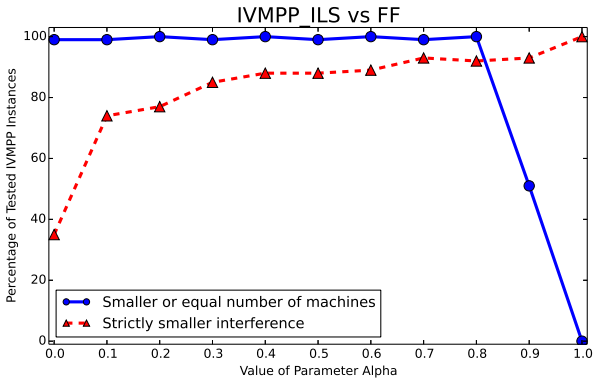


Figure 5.2: IVMPP_ILS vs FF

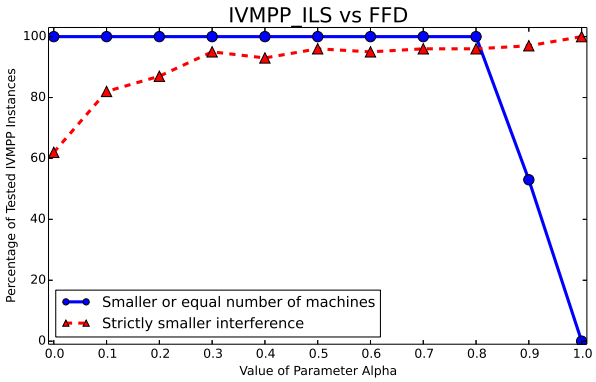


Figure 5.3: IVMPP_ILS vs FFD

Results of the comparison between IVMPP_ILS and heuristics FF, FFD, BF, BFD,

WF, and WFD are presented in Figures 5.2 to 5.7, respectively. As expected, when α was set to 0.00 or 1.00, IVMPP_ILS prioritizes one or another objective. More specifically, when α was equal to 1.0, IVMPP_ILS always achieved, in comparison with all heuristics, a strictly smaller sum of interference levels, though using a greater number of physical machines. This happens because, in such configuration, IVMPP_ILS allocates each application to a different physical machine.

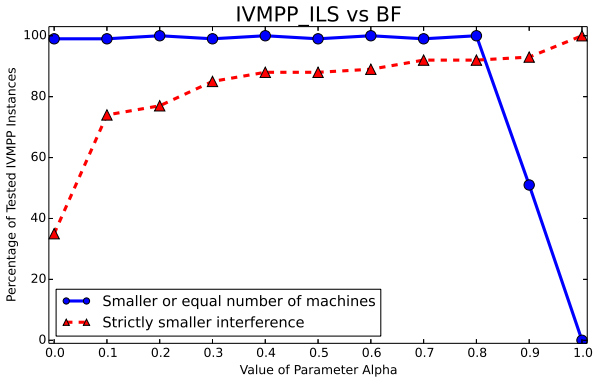


Figure 5.4: IVMPP_ILS vs BF

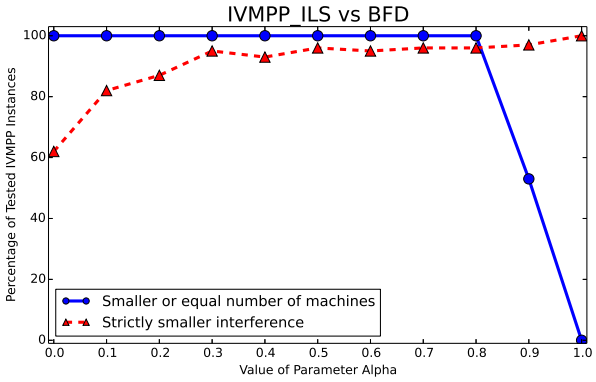


Figure 5.5: IVMPP_ILS vs BFD

On the other hand, when α was set to 0.00, IVMPP_ILS presented, for all heuristics, the worst result concerning interference minimization. However, even in that case, our solution outperformed the heuristics in, at least, 32% of instances (result for WF, as shown in Figure 5.6). This behaviour can be explained by the strategy used to construct initial solutions. As previously described in Section 5.2, an initial solution is generated by co-allocating applications with complementary access profiles and complementary amount of requested resources. As a consequence, the initial solution tends to present a low sum of interference levels and number of physical machines, as well. Thus, even when IVMPP_ILS seeks only for the minimization of used physical machines, the initial solution may present small interference levels.

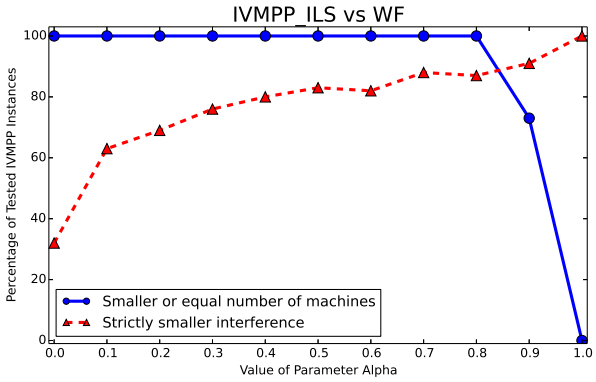


Figure 5.6: IVMPP_ILS vs WF

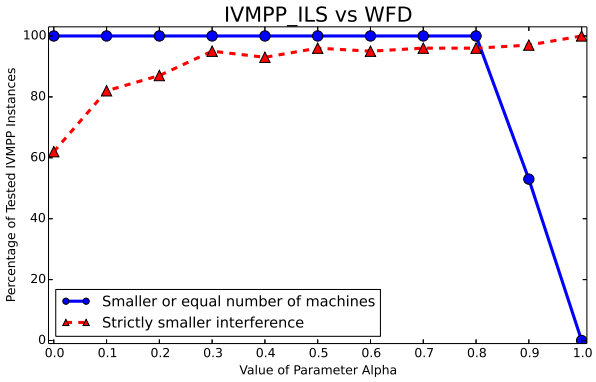


Figure 5.7: IVMPP_ILS vs WFD

Considering all experiments, the best trade-off between reducing interference and number of used machines was achieved by our proposal when α was equal to 0.70. In this case, our proposal has always achieved a smaller or equal number of used machines, besides reaching, at least, a smaller interference in 88% of all evaluated instances, as shown in Figure 5.6. In other words, this value of α enabled IVMPP_ILS to reach, at the same time, the highest values for both metrics in all experiments, i.e., IVMPP_ILS achieved (i) a strictly smaller sum of interference levels and (ii) a smaller or equal number of physical machines than heuristics in 88% and 100% of tested cases, respectively.

However, it is worth mentioning that the heuristic achieved the best interference levels only when it also used a greater number of physical machines than our solutions (see Figure 5.8). This behaviour is expected since interference can really be reduced by allocating applications to more physical machines. In those cases, the heuristic penalized the minimization of physical machines to achieve a lower sum of interference levels. In short, in those cases, heuristic solutions were not better than our solutions.

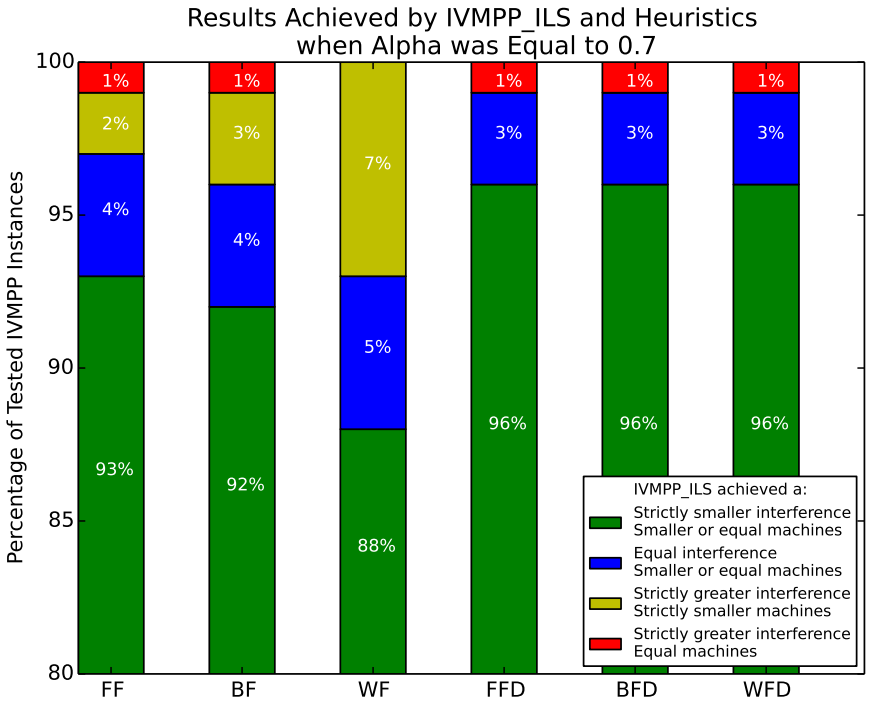


Figure 5.8: Results achieved by IVMPP_ILS and heuristcs when alpha was equal to 0.7

In only 2 instances, which represents 2% of all tested cases, all heuristics (except for WF) indeed outperformed our solution by achieving a smaller sum of interference levels while using the same number of machines indicated by our proposal. As shown in Table 5.3, heuristics FF and BF, for instance "IVMPP_5.5", reached a smaller sum of interference

levels than IVMPP_ILS. This same behavior can be observed for heuristics FFD, BFD and WFD when solving the instance "IVMPP_40.7". Even so, the difference between the sum of interference levels was smaller than 1%. More specifically, the difference was equal to 0.03% and 0.95% for instances "IVMPP_5.5" and "IVMPP_40.7", respectively.

			Sum of Interference Levels		
Instance	**Number of Applications**	**Heuristics Outperformed IVMPP_ILS**	**IVMPP_ILS**	**Heuristics**	**Number of Used Physical Machines**
IVMPP_5.5	5	FF, BF	13.27%	13.24%	2
IVMPP_40.7	40	FFD, BFD, WFD	113.02%	112.07%	14

Table 5.3: Cases where heuristics outperformed IVMPP_ILS

Those results indicate that our proposal was able to reduce the sum of interference levels, while using a small, perhaps minimal, number of physical machines. In addition, the results showed that the cloud provider can use our strategy to dynamically adjust the behavior of its virtual machine placement policy by varying the value of parameter α. Thus, the cloud provider can easily prioritizes one or another objective according to its needs.

5.3.2.2 Experiments in a Real Scenario

By using the selected value for the parameter α (0.70), we conducted a second phase of experiments to verify, in a real scenario, the efficiency of IVMPP_ILS in reducing the sum of interference levels, while keeping the same number of physical machines given by the greedy heuristics.

Thereby, we selected five instances where the number of physical machines given by our solution and heuristics was exactly the same. Thus, we could accomplish a fair comparison between the tested strategies. For each instance, we executed applications in accordance with the placement determined by IVMPP_ILS and by the heuristic that presented the lower sum of interference levels in offline experiments. Next, we calculated the sum of interference levels that our proposal and the heuristic achieved in the real scenario. These experiments were executed in a set of physical machines whose configurations were detailed in Section 3.2.2.

Results show that our strategy, even using the same number of physical machines of the other heuristic, reduced the sum of interference levels up to 41%, as presented in

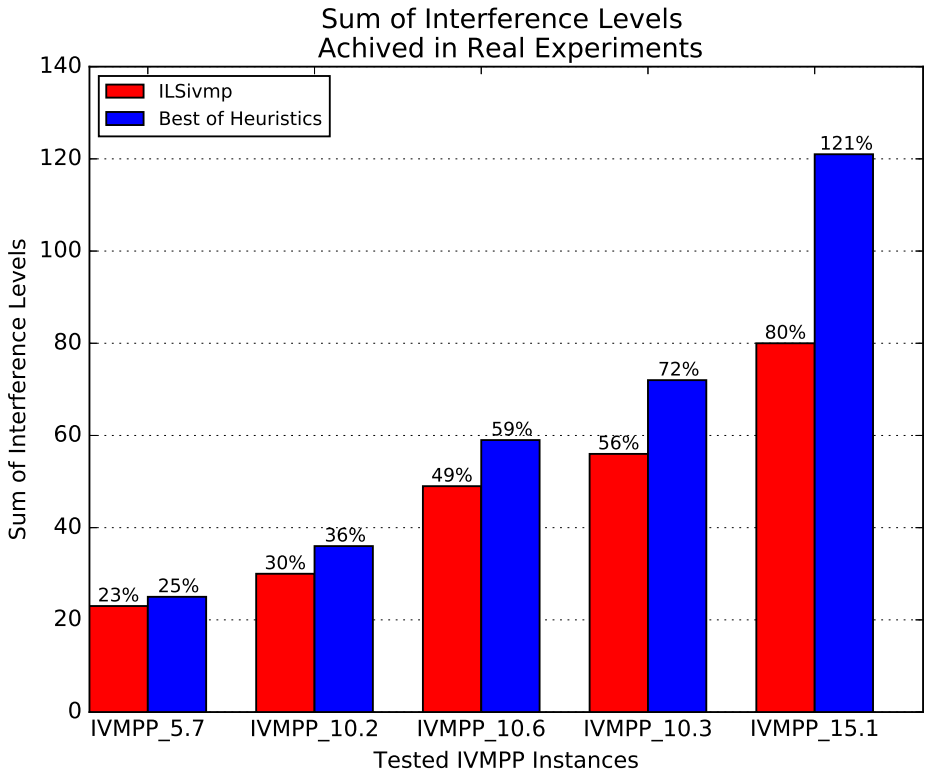


Figure 5.9: Sum of interference levels achieved in a real scenario

Figure 5.9. Moreover, it is worth mentioning that the difference between the predicted sum of interference levels and the one achieved in real experiments was, in average, around 8%.

Notice that the highest difference between the sum of interference levels achieved by our solution and the one reached by heuristics occurred when they solved the instance "IVMPP_15.1" whose details are described in Table 5.4. In such case, IVMPP_ILS reached a sum of interference levels 41.25% smaller than FF, the best greedy heuristic in this case.

IVMPP_ILS outperformed FF in that case because it co-located applications with complementary access profiles, as can be seen in Figure 5.10. For example, our solution placed "PTRANS.I1.P6", a high access rate application, in a dedicated physical machine ("PM2"), thus, avoiding a possible cross-interference. On the other hand, heuristic FF, unlike our proposal, allocated "PTRANS.I1.P6" together with "PTRANS.I1.P4" and "DGEMM.I1.P2", leading to an interference level equal to 32.83%. This high interference level stems from the fact that those applications put a high pressure on the three shared resources, as can be verified in Table 5.4.

Besides that, FF achieved a high interference level in the physical machine "PM1", around 46%, because it co-located applications that resulted in accumulated access scores

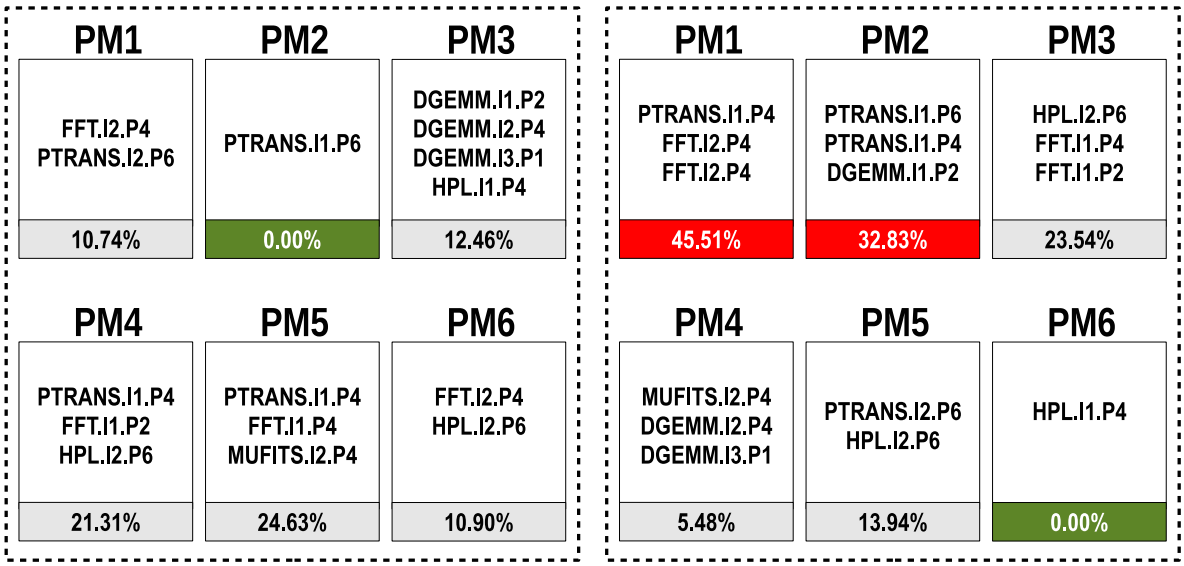
		Individual Access			Required Resources	
App. Label	**Number of VM**	**SLLC**	**DRAM**	**NET**	**CPU**	**Memory**
PTRANS.I1.4	4	0.1362	0.0908	0.1938	0.33	0.33
FFT.I2.4	4	0.0703	0.1643	0.5168	0.33	0.33
PTRANS.I1.6	6	0.1828	0.2135	0.3216	0.50	0.50
FFT.I2.4	4	0.0703	0.1643	0.5168	0.33	0.33
PTRANS.I1.4	4	0.1362	0.0908	0.1938	0.33	0.33
HPL.I2.6	6	0.0277	0.0566	0.0192	0.50	0.50
DGEMM.I1.2	2	0.0043	0.0090	0.0000	0.17	0.17
FFT.I1.4	4	0.0680	0.1694	0.4931	0.33	0.33
MUFITS.I1.4	4	0.0245	0.0004	0.0058	0.33	0.33
FFT.I2.4	4	0.0410	0.0856	0.3058	0.17	0.17
DGEMM.I2.4	4	0.0067	0.0158	0.0000	0.33	0.33
DGEMM.I3.1	1	0.0031	0.0045	0.0000	0.08	0.08
PTRANS.I2.6	6	0.0177	0.0002	0.1113	0.50	0.50
HPL.I2.6	6	0.0277	0.0566	0.0192	0.50	0.50
HPL.I1.4	4	0.0180	0.0414	0.0137	0.33	0.33

Table 5.4: Instance IVMPP_15.1

equal to 0.28, 0.42 and 1.23 for SLLC, DRAM and virtual network, respectively. Conversely, our solution avoided to allocate, in the same physical machine, two applications "FFT.I2.P4" because this application presents the highest access rate for the virtual network.

Furthermore, we could observe that the difference between the sum of interference levels achieved by our solution and the greedy heuristics tended to increase with the rising of the number of applications. For example, our solution achieved a sum of interference levels 2% smaller than the heuristics solutions when solving instance "IVMPP_5.7". On the other hand, for instance "IVMPP_15.1", which has 10 times more applications than "IVMPP_5.7', this difference raised to 41%. In fact, as a higher number of applications incurs in a larger amount of placement combinations and our solution explores several distinct combinations of co-locations, it is expected, for those cases, that IVMPP_ILS achieves smaller sums of interference levels solutions than the ones greedily found by the heuristics.

Those results allowed to confirm conclusions drawn from offline experiments. IVMPP_ILS was really able to reduce the level of interference experienced by HPC applications without increasing the number of physical machines needed to allocate them in cloud environments.



(a) IVMPP_ILS (b) FF

Figure 5.10: Interference levels break down per physical machine for placement given by IVMPP_ILS and FF when solving IVMPP_15.1

5.3.3 Comparing IVMPP_ILS with an Exact Approach

As presented in the previous section, IVMPP_ILS achieved good solutions in a reasonable execution time, spending less than 15 seconds to solve each instance, when compared with other heuristics from the related literature. In spite of those satisfactory results, there is no guarantee that our proposal has achieved the best solution for any of the tested IVMPP instances, since metaheuristics, although can often find good solutions with less computational effort than exact approaches, do not guarantee that a globally optimal solution can be found. So, to have another parameter of solution quality, IVMPP_ILS was also compared with IVMPP_EXACT, a naive exact procedure for solving IVMPP, in a sample of small instances.

IVMPP_EXACT enumerates all possible solutions and selects the one with the smallest cost. This sort of procedure, also known as exhaustive or direct search, evaluates each possible solution of a discrete problem to determine the optimal one. As this exact approach executes an exhaustive search on the space of possible solutions, the computational time of this method grows exponentially with the size of instance being solved. In the context of IVMPP, the size of instance is related to the number of applications to be allocated in the cloud environment. Thus, the greatest the number of applications in the instance, the higher the execution time of IVMPP_EXACT is.

Due to its computational complexity, IVMPP_EXACT was just used to solve small

Number of App.	Instance	IVMPP_EXACT		IVMPP_ILS		Gap
		Cost	Time (s)	Cost	Time (s)	
5	IVMPP_5.1	0.123759	0	0.123788	0	0.02%
	IVMPP_5.2	0.128015	0	0.128015	0	0.00%
	IVMPP_5.3	0.128016	0	0.128016	0	0.00%
	IVMPP_5.4	0.128427	0	0.128427	0	0.00%
	IVMPP_5.5	0.127375	0	0.127398	0	0.02%
	IVMPP_5.6	0.132018	0	0.132083	0	0.05%
	IVMPP_5.7	0.132136	0	0.132205	0	0.05%
	IVMPP_5.8	0.130486	0	0.130486	0	0.00%
	IVMPP_5.9	0.126877	0	0.126885	0	0.01%
	IVMPP_5.10	0.135305	0	0.135305	0	0.00%
6	IVMPP_6.1	0.107733	0	0.107735	0	0.00%
	IVMPP_6.2	0.157338	0	0.157345	0	0.00%
	IVMPP_6.3	0.117550	0	0.117617	0	0.06%
	IVMPP_6.4	0.110082	0	0.110082	0	0.00%
	IVMPP_6.5	0.105597	0	0.105597	0	0.00%
	IVMPP_6.6	0.120686	0	0.120686	0	0.00%
	IVMPP_6.7	0.153522	0	0.153522	0	0.00%
	IVMPP_6.8	0.152300	0	0.152312	0	0.01%
	IVMPP_6.9	0.121506	0	0.121684	0	0.15%
	IVMPP_6.10	0.112187	0	0.112190	0	0.00%
7	IVMPP_7.1	0.134135	16	0.134153	0	0.01%
	IVMPP_7.2	0.139690	16	0.139690	0	0.00%
	IVMPP_7.3	0.141633	16	0.141645	0	0.01%
	IVMPP_7.4	0.132102	16	0.132105	0	0.00%
	IVMPP_7.5	0.173459	16	0.173459	0	0.00%
	IVMPP_7.6	0.137693	16	0.137714	0	0.02%
	IVMPP_7.7	0.129884	16	0.129889	0	0.00%
	IVMPP_7.8	0.106974	16	0.106994	0	0.02%
	IVMPP_7.9	0.134610	16	0.134610	0	0.00%
	IVMPP_7.10	0.135761	16	0.135770	0	0.01%
8	IVMPP_8.1	0.152632	111	0.152638	0	0.00%
	IVMPP_8.2	0.125171	111	0.125364	0	0.15%
	IVMPP_8.3	0.124114	111	0.124148	0	0.03%
	IVMPP_8.4	0.114537	110	0.114578	0	0.04%
	IVMPP_8.5	0.119423	110	0.119555	0	0.11%
	IVMPP_8.6	0.117526	112	0.117544	0	0.02%
	IVMPP_8.7	0.156977	112	0.157014	0	0.02%
	IVMPP_8.8	0.117246	112	0.117263	0	0.01%
	IVMPP_8.9	0.154842	112	0.154864	0	0.01%
	IVMPP_8.10	0.130006	110	0.130008	0	0.00%

Table 5.5: Results achieved by IVMPP_ILS and IVMPP_EXACT when solving a set of instances

instances which could be solved within 12 hours time limit. Therefore, we considered instances with 5, 6, 7 and 8 applications, where instances with 6 to 8 applications were exclusively created for this analysis and were not considered in our previous experiments. To solve instances with 8 applications, we implemented a parallel version of IVMPP_EXACT that was executed with 12 processors. It is worth mentioning that we have tried to solve instances with 9 applications. However, even using 12 processors, the exact procedure was not able to solve this kind of instance in less than 24 hours.

As can be seen in Table 5.5, IVMPP_ILS obtained good solutions with small gaps, around 0.02% in average. Moreover, for some instances ("IVMPP_7.9" and "IVMPP_5.2", for example), our solution and the exact procedure achieved exactly the same cost. Thus, for that cases, IVMPP_ILS was able to reach the optimal solution for the problem. Those experiments were repeated 10 times and the coefficient of variation, for all cases, was less than 0.18%.

At last, we acknowledge that this evaluation is not enough to assess the capability of our proposal on reaching optimal or, at least, near-optimal solutions. However, we claim that those results, together with the previous experiments, indicate that our solution is indeed capable of giving good solutions for the IVMPP.

Chapter 6

Conclusions and Future Work

This chapter describes the main results and contributions of this work. In Section 6.1, we highlight the main findings of this thesis, while we point out in Section 6.2 some interesting directions for future research in this topic area.

6.1 Concluding Remarks

In this work, we defined the Interference-aware Virtual Machine Placement Problem for Small-scale HPC Applications in Clouds (IVMPP). This problem aims to minimize, simultaneously, (i) the interference experienced by small-scale HPC applications executed in a same physical machine and (ii) the number of physical machines used to allocate them. Besides presenting a mathematical formulation for the problem, we introduced a strategy based on the Iterated Local Search (ILS) framework to solve it. This strategy, called IVMPP_ILS, employs a multistart approach where new solutions are repeatedly submitted to the ILS as a means to create a more effective searching process.

To estimate interference for a set of co-located applications, our VMP strategy uses a quantitative and multivariate interference prediction model that was also proposed in this work. This model considers the amount and similarities of concurrent access to three shared and non-sliceable resources (SLLC, DRAM and virtual network) and the number of co-located applications. Because of the multivariate nature of the cross-interference problem, we generated this model by applying a multivariate statistical technique in a synthetic interference dataset. This dataset was created through interference experiments accomplished with applications that put distinct access pressure on shared resources.

The prediction model was validated by using a petroleum reservoir simulator, a seismic

migration method and applications from the widely adopted HPCC benchmark. These real-life HPC applications were executed with instances of distinct sizes and characteristics, and experiments were accomplished by considering different co-location schemes and number of applications. Results showed that our model achieved a median prediction error equal to 3.59% and, for 93% of all tested cases, it reached a prediction error less than 10%. These results pointed out that our model, though it was built from a synthetic interference dataset, was capable of predicting interference for a set of real small-scale HPC applications running in distinct circumstances. Moreover, our experiments also indicated that solutions based just on (i) number of running virtual machines, (ii) Dwarfs classes or (iii) SLLC access contention may not be suitable for determining interference.

By using a set of instances created from the validation workload, we carried out experiments to compare our proposal with the most common heuristics used to minimize the number of active machines in clouds. At first, we accomplished several offline experiments to calibrate our proposal concerning its ability to find a good trade-off between reducing, at the same time, the interference and number of used machines. After that, we conducted real experiments to confirm, in practice, the conclusions drawn in offline experiments. Results showed that our strategy reduced interference up to 41%, while keeping the same number of physical machines given by those heuristics. IVMPP_ILS outperformed other heuristics because, even using the minimal number of physical machines, it was able to co-locate applications with complementary access profiles.

In addition, we evaluated the optimality of solutions given by IVMPP_ILS. We compared it with an exact procedure that is able to find the optimal solution for an IVMPP instance. This experimental evaluation revealed that our strategy achieved good solutions since the gap was, in average, around 0.02%. Although this experimental evaluation was accomplished with small IVMPP instances, these results corroborate the claim that our proposal is able to find good solutions for the IVMPP.

6.2 Future Work

In this section, we present some promising future directions derived from the contributions of this thesis.

1. **Consider other shared resources**: our proposed model considers concurrent accesses to SLLC, DRAM and virtual network to estimate the level of interference experienced by co-located applications. However, it would be interesting to extend

the proposed model so that it takes into account the negative impact that concurrent access to other shared resources has in interference. For example, the model could consider the amount of simultaneous access to disk since the concurrent access to this shared resource may have a devastating influence in cross-application interference [33].

2. **Evaluate access contention with other metrics**: to measure the amount of access to DRAM and virtual network, we used, in this work, the number of references per second and the amount of bytes transmitted per second, respectively. We adopted these metrics because they allow to capture the pressure that co-located applications put in those shared resources. Although those metrics have proven to be suitable for predicting interference, perhaps other metrics, such as the total amount of allocated memory or the number of packets transmitted, for example, could reveal another perspective of the access contention degree imposed by those shared resources.

3. **Incorporate hardware characteristics in the prediction model**: all experiments described in this thesis were executed in the same hardware configuration, that is, in the same physical machine model. So, we did not conduct any investigation to verify whether interference varies in distinct hardware configurations. The negative impact caused by access contention may change depending on issues like the size of SLLC, number of DRAM channels, memory frequency and so on. To incorporate such additional information into the prediction model, it could be used, for example, online machine learning techniques[37]. Thus, the model coefficients could be dynamically adjusted concerning those specific characteristics in the cloud environment.

4. **Online approach:** to decide the placement of virtual machines, our strategy needs to know in advance the set of applications that should be allocated in the cloud. Although this offline approach can be used in practice by accumulating a given number of requests before performing placement decisions, an online approach, which allocates virtual machines as they arrive, could enhance the contribution of this work. To propose that strategy, the online approach should be able to determine in what extent the live migration of virtual machines would affect the execution of small-scale HPC applications.

5. **Communication through physical network:** due to the poor scalability of HPC applications in clouds, we decided to focus our research in small-scale HPC

applications, that is, applications that could be entirely placed in a physical machine. But, depending on the level of interference, an application could benefit more of using the physical network than of being fully allocated to a physical machine. Thus, our proposed strategy could be expanded to evaluate the trade-off between reducing the level of interference and the cost of using the physical network. This improvement could enhance the solution so that it was able to deal with medium and high-scale HPC applications also.

Bibliography

[1] EViews. http://www.eviews.com. Last accessed in February 2018.

[2] HPC Challenge Benchmark. http://icl.cs.utk.edu/hpcc/. Last accessed in February 2018.

[3] IBM SPSS Statistics. http://www.ibm.com/SPSS/Statistics. Last accessed in February 2018.

[4] Minitab. http://www.minitab.com/. Last accessed in February 2018.

[5] MPBench. http://icl.cs.utk.edu/llcbench/mpbench.html. Last accessed in February 2018.

[6] MUFITS Reservoir Simulation Software. http://www.mufits.imec.msu.ru/. Last accessed in February 2018.

[7] OProfile. http://oprofile.sourceforge.net. Last accessed in February 2018.

[8] Perf Monitoring Tool. https://perf.wiki.kernel.org/index.php/Main_Page. Last accessed in February 2018.

[9] Performance Application Programming Interface (PAPI). http://icl.utk.edu/papi/. Last accessed in February 2018.

[10] STREAM: Sustainable Memory Bandwidth in High Performance Computers. https://www.cs.virginia.edu/stream/. Last accessed in February 2018.

[11] Abdelkhalek, R.; Calandra, H.; Coulaud, O.; Roman, J.; Latu, G. Fast Seismic Modeling and Reverse Time Migration on a GPU Cluster. In *International Conference on High Performance Computing & Simulation.* (2009), IEEE, pp. 36–43.

[12] Albericio, J.; Ibáñez, P.; Viñals, V.; Llabería, J. M. The Reuse Cache: Downsizing the Shared Last-level Cache. In *Proceedings of the 46th Annual International Symposium on Microarchitecture* (2013), ACM, pp. 310–321.

[13] Alves, M.; Drummond, L. Análise de Desempenho de um Simulador de Reservatórios de Petróleo em um Ambiente de Computação em Nuvem. In *XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)* (2014).

[14] Alves, M.; Pestana, R.; Silva, R.; Drummond, L. Accelerating Pre-stack Kirchhoff Time Migration by Manual Vectorization. *Concurrency and Computation: Practice and Experience (online)* (2016).

[15] Andreolini, M.; Casolari, S.; Colajanni, M.; Messori, M. Dynamic Load Management of Virtual Machines in Cloud Architectures. In *International Conference on Cloud Computing* (2009), Springer, pp. 201–214.

[16] Anzt, H.; Dongarra, J.; Gates, M.; Kurzak, J.; Luszczek, P.; Tomov, S.; Yamazaki, I. Bringing High Performance Computing to Big Data Algorithms. In *Handbook of Big Data Technologies*. Springer, 2017, pp. 777–806.

[17] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R. H.; Konwinski, A.; Lee, G.; Patterson, D. A.; Rabkin, A.; Stoica, I.; Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing. Tech. rep., UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[18] Atif, M.; Kobayashi, R.; Menadue, B. J.; Lin, C. Y.; Sanderson, M.; Williams, A. Breaking HPC Barriers with the 56GbE Cloud. *Procedia Computer Science 93* (2016), 3–11.

[19] Babu, K. R.; Samuel, P. Virtual Machine Placement for Improved Quality in IaaS Cloud. In *Fourth International Conference on Advances in Computing and Communications (ICACC)* (2014), IEEE, pp. 190–194.

[20] Baiocchi, G.; Distaso, W. GRETL: Econometric Software for the GNU Generation. *Journal of Applied Econometrics 18*, 1 (2003), 105–110.

[21] Basto, D. T. Interference Aware Scheduling for Cloud Computing. Master's thesis, Universidade do Porto, 2015.

[22] Beloglazov, A.; Abawajy, J.; Buyya, R. Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. *Future Generation Computer Systems 28*, 5 (2012), 755–768.

[23] Benedict, S.; Rejitha, R.; Preethi, C.; Bright, C. B.; Judyfer, W. Energy Analysis of Code Regions of HPC Applications using Energy Analyzer Tool.

International Journal of Computational Science and Engineering 14, 3 (2017), 267–278.

[24] Bhardwaj, D.; Phadke, S.; Yerneni, S. On Improving Performance of Migration Algorithms using MPI and MPI-IO. In *Expanded Abstracts, Society of Exploration Geophysicists* (2000).

[25] Bobroff, N.; Kochut, A.; Beaty, K. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *10th International Symposium on Integrated Network Management* (2007), IEEE, pp. 119–128.

[26] Boyer, V.; El Baz, D.; Elkihel, M. Solving Knapsack Problems on GPU. *Computers & Operations Research 39*, 1 (2012), 42–47.

[27] Chen, L.; Patel, S.; Shen, H.; Zhou, Z. Profiling and Understanding Virtualization Overhead in Cloud. In *44th International Conference on Parallel Processing (ICPP)* (2015), IEEE, pp. 31–40.

[28] Chen, L.; Shen, H.; Platt, S. Cache Contention Aware Virtual Machine Placement and Migration in Cloud Datacenters. In *24th International Conference on Network Protocols (ICNP)* (2016), IEEE, pp. 1–10.

[29] Chi, R.; Qian, Z.; Lu, S. Be a Good Neighbour: Characterizing Performance Interference of Virtual Machines under Xen Virtualization Environments. In *International Conference on Parallel and Distributed Systems (ICPADS)* (2014), IEEE, pp. 257–264.

[30] da Silva, R. A. P.; Alves, M. M.; Bentes, C. B.; de Assumpção Drummond, L. M. Vetorização e Análise de Algoritmos Paralelos para a Migração Kirchhoff Pré-empilhamento em Tempo. In *XVIII Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)* (2017).

[31] Dali, N.; Bouamama, S. Different Parallelism Levels using GPU for Solving Max-CSPs with PSO. In *Congress on Evolutionary Computation (CEC)* (2017), IEEE, pp. 2070–2077.

[32] Dongarra, J.; Luszczek, P. HPC Challenge: Design, History, and Implementation Highlights. *Contemporary High Performance Computing: From Petascale Toward Exascale* (2013).

[33] Dorier, M.; Antoniu, G.; Ross, R.; Kimpe, D.; Ibrahim, S. CALCioM: Mitigating I/O Interference in HPC Systems through Cross-application Coordination. In *28th International Parallel and Distributed Processing Symposium* (2014), IEEE, pp. 155–164.

[34] El-Gazzar, R.; Hustad, E.; Olsen, D. H. Understanding Cloud Computing Adoption Issues: A Delphi Study Approach. *Journal of Systems and Software 118* (2016), 64–84.

[35] Evangelinos, C.; Hill, C. Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-ocean Climate Models on Amazon EC2. In *Cloud Computing and Its Applications* (2008), Computation Institute and the National Center for Data Mining at UIC, pp. 2–34.

[36] Expósito, R. R.; Taboada, G. L.; Ramos, S.; Touriño, J.; Doallo, R. Performance Analysis of HPC Applications in the Cloud. *Future Generation Computer Systems 29*, 1 (2013), 218–229.

[37] Fontenla-Romero, Ó.; Guijarro-Berdiñas, B.; Martinez-Rego, D.; Pérez-Sánchez, B.; Peteiro-Barral, D. Online Machine Learning. *Efficiency and Scalability Methods for Computational Intellect 27* (2013).

[38] Gentzsch, W.; Yenier, B. The UberCloud HPC Experiment: Compendium of Case Studies. Tech. rep., Tabor Communications, 2013.

[39] Goldbarg, M. C.; Luna, H. P. L. *Otimização Combinatória e Programação Linear: Modelos e Algoritmos*, vol. 1. Editora Campus, Rio de Janeiro, 2000.

[40] Guan, W.; Qiao, C.; Zhang, H.; Zhang, C.-S.; Zhi, M.; Zhu, Z.; Zheng, Z.; Ye, W.; Zhang, Y.; Hu, X.; Li, Z.; Feng, C.; Xu, Y.; Xu, J. On Robust and Efficient Parallel Reservoir Simulation on Tianhe-2. In *SPE Reservoir Characterisation and Simulation Conference and Exhibition* (2015), Society of Petroleum Engineers.

[41] Gupta, A. *Techniques for Efficient High Performance Computing in the Cloud*. PhD thesis, University of Illinois, 2014.

[42] Gupta, A.; Faraboschi, P.; Gioachin, F.; Kale, L. V.; Kaufmann, R.; Lee, B.-s.; March, V.; Milojicic, D.; Suen, C. H. Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. *IEEE Transactions on Cloud Computing 7161* (2014), 1–1.

[43] Gupta, A.; Kale, L. V.; Milojicic, D.; Faraboschi, P.; Balle, S. M. HPC-aware VM Placement in Infrastructure Clouds. In *International Conference on Cloud Engineering (IC2E)* (2013), IEEE, pp. 11–20.

[44] Gupta, A.; Milojicic, D. Evaluation of HPC Applications on Cloud. In *Open Cirrus Summit (OCS)* (2011), IEEE, pp. 22–26.

[45] Hair, J. F.; Black, W. C.; Babin, B. J.; Anderson, R. E.; Tatham, R. L. *Multivariate Data Analysis (Vol. 6)*. Upper Saddle River, NJ: Pearson Prentice Hall, 2006.

[46] Hamid, N. A. W. A.; Coddington, P. Comparison of MPI Benchmark Programs on Shared Memory and Distributed Memory Machines (Point-to-Point Communication). *The International Journal of High Performance Computing Applications 24*, 4 (2010), 469–483.

[47] Hassan, H. A.; Mohamed, S. A.; Sheta, W. M. Scalability and Communication Performance of HPC on Azure Cloud. *Egyptian Informatics Journal 17*, 2 (2016), 175–182.

[48] Heiser, G. The Role of Virtualization in Embedded Systems. In *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems* (2008), ACM, pp. 11–16.

[49] Hill, Z.; Humphrey, M. A Quantitative Analysis of High Performance Computing with Amazon EC2 Infrastructure: The Death of the Local Cluster? In *International Conference on Grid Computing* (2009), IEEE, pp. 26–33.

[50] Jackson, K. R.; Ramakrishnan, L.; Muriki, K.; Canon, S.; Cholia, S.; Shalf, J.; Wasserman, H. J.; Wright, N. J. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Second International Conference on Cloud Computing Technology and Science (CloudCom)* (2010), IEEE, pp. 159–168.

[51] Jersak, L. C.; Ferreto, T. Performance-aware Server Consolidation with Adjustable Interference Levels. In *Proceedings of the 31st Annual Symposium on Applied Computing* (2016), ACM, pp. 420–425.

[52] Jin, H.; Qin, H.; Wu, S.; Guo, X. CCAP: A Cache Contention-aware Virtual Machine Placement Approach for HPC Cloud. *International Journal of Parallel Programming 43*, 3 (2015), 403–420.

[53] Koh, Y.; Knauerhase, R.; Brett, P.; Bowman, M.; Wen, Z.; Pu, C. An Analysis of Performance Interference Effects in Virtual Environments. In *International Symposium on Performance Analysis of Systems & Software (ISPASS)* (2007), IEEE, pp. 200–209.

[54] Lelli, J.; Faggioli, D.; Cucinotta, T.; Lipari, G. An Experimental Comparison of Different Real-time Schedulers on Multicore Systems. *Journal of Systems and Software 85*, 10 (2012), 2405–2416.

[55] Li, Y.; Tang, X.; Cai, W. Dynamic Bin Packing for On-demand Cloud Resource Allocation. *Transactions on Parallel and Distributed Systems 27*, 1 (2016), 157–170.

[56] Liu, H.; Li, B.; Liu, H.; Tong, X.; Liu, Q.; Wang, X.; Liu, W. The Issues of Prestack Reverse Time Migration and Solutions with Graphic Processing Unit Implementation. *Geophysical Prospecting 60*, 5 (2012), 906–918.

[57] Liu, W.; Nemeth, T.; Loddoch, A.; Stefani, J.; Ergas, R.; Zhuo, L.; Volz, B.; Pell, O.; Huggett, J. Anisotropic Reverse-time Migration using Co-processors. In *SEG Annual Meeting* (2009), Society of Exploration Geophysicists.

[58] Lourenço, H. R.; Martin, O. C.; Stützle, T. *Iterated Local Search*. Springer, 2003.

[59] Lu, B.; Wheeler, M. F. Iterative Coupling Reservoir Simulation on High Performance Computers. *Petroleum Science 6*, 1 (2009), 43–50.

[60] Luszczek, P.; Dongarra, J. J.; Koester, D.; Rabenseifner, R.; Lucas, B.; Kepner, J.; McCalpin, J.; Bailey, D.; Takahashi, D. Introduction to the HPC Challenge Benchmark Suite. *Lawrence Berkeley National Laboratory* (2005).

[61] Madhumathi, R.; Radhakrishnan, R.; Balagopalan, A. Dynamic Resource Allocation in Cloud using Bin-packing Technique. In *International Conference on Advanced Computing and Communication Systems* (2015), IEEE, pp. 1–4.

[62] Maleki, S.; Gao, Y.; Garzaran, M. J.; Wong, T.; Padua, D. A. An Evaluation of Vectorizing Compilers. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)* (2011), IEEE, pp. 372–382.

[63] Martello, S.; Toth, P. Lower Bounds and Reduction Procedures for the Bin Packing Problem. *Discrete Applied Mathematics 28*, 1 (1990), 59–70.

[64] Martí, R. Multi-start Methods. *International Series in Operations Research and Management Science* (2003), 355–368.

[65] Mason, C. H.; Perreault Jr, W. D. Collinearity, Power, and Interpretation of Multiple Regression Analysis. *Journal of Marketing Research* (1991), 268–280.

[66] Mehrotra, P.; Djomehri, J.; Heistand, S.; Hood, R.; Jin, H.; Lazanoff, A.; Saini, S.; Biswas, R. Performance Evaluation of Amazon Elastic Compute Cloud for NASA High-performance Computing Applications. *Concurrency and Computation: Practice and Experience 28*, 4 (2016), 1041–1055.

[67] Mishra, M.; Sahoo, A. On Theory of VM Placement: Anomalies in Existing Methodologies and their Mitigation using a Novel Vector Based Approach. In *International Conference on Cloud Computing (CLOUD)* (2011), IEEE, pp. 275–282.

[68] Mury, A. R.; Schulze, B.; Licht, F. L.; de Bona, L. C.; Ferro, M. A Concurrency Mitigation Proposal for Sharing Environments: An Affinity Approach Based on Applications Classes. In *Intelligent Cloud Computing*. Springer, 2014, pp. 26–45.

[69] Nasim, R.; Taheri, J.; Kassler, A. Optimizing Virtual Machine Consolidation in Virtualized Datacenters Using Resource Sensitivity. In *8th International Conference on Cloud Computing Technology and Science (cloudCom2016)* (2016), IEEE.

[70] Netto, M. A.; Calheiros, R. N.; Rodrigues, E. R.; Cunha, R. L.; Buyya, R. HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *ACM Computing Surveys 1*, 1 (2017).

[71] Ngo, T. H. D.; La Puente, C. The Steps to Follow in a Multiple Regression Analysis. In *Proceedings of the SAS Global Forum Conference* (2012), Citeseer.

[72] Otto, C.; Kempka, T. Prediction of Steam Jacket Dynamics and Water Balances in Underground Coal Gasification. *Energies 10*, 6 (2017), 739.

[73] Peaceman, D. W. *Fundamentals of Numerical Reservoir Simulation*. Elsevier, 2000.

[74] Pires, F. L.; Barán, B. A Virtual Machine Placement Taxonomy. In *15th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2015), IEEE/ACM, pp. 159–168.

[75] Pires, L. F.; Baran, B. Virtual Machine Placement Literature Review. Tech. rep., Polytechnic School, University of Asuncion, 2014.

[76] Popiolek, P. F.; Mendizabal, O. M. Monitoring and Analysis of Performance Impact in Virtualized Environments. *Journal of Applied Computing Research 2*, 2 (2013), 75–82.

[77] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[78] Rameshan, N.; Navarro, L.; Monte, E.; Vlassov, V. Stay-away, Protecting Sensitive Applications From Performance Interference. In *Proceedings of the 15th International Middleware Conference* (2014), ACM, pp. 301–312.

[79] Rastogi, R.; Phadke, S. Optimal Aperture Width Selection and Parallel Implementation of Kirchhoff Migration Algorithm. In *Fourth International Conference and Exposition of the Society of Petroleum Geophysicists* (2002), Citeseer, pp. 7–9.

[80] Ray, B. R.; Chowdhury, M.; Atif, U. Is High Performance Computing (HPC) Ready to Handle Big Data? In *International Conference on Future Network Systems and Security* (2017), Springer, pp. 97–112.

[81] Roloff, E.; Diener, M.; Gaspary, L. P.; Navaux, P. O. HPC Application Performance and Cost Efficiency in the Cloud. In *25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)* (2017), IEEE, pp. 473–477.

[82] Sadooghi, I.; Martin, J. H.; Li, T.; Brandstatter, K.; Maheshwari, K.; de Lacerda Ruivo, T. P. P.; Garzoglio, G.; Timm, S.; Zhao, Y.; Raicu, I. Understanding the Performance and Potential of Cloud Computing for Scientific Applications. *Transactions on Cloud Computing 5*, 2 (2017), 358–371.

[83] Schulz, C. Efficient Local Search on the GPU-investigations on the Vehicle Routing Problem. *Journal of Parallel and Distributed Computing 73*, 1 (2013), 14–31.

[84] Shirvani, M. H.; Ghojoghi, A. Server Consolidation Schemes in Cloud Computing Environment: A Review. *European Journal of Engineering Research and Science 1*, 3 (2016).

[85] Silva, J.; Boeres, C.; Drummond, L.; Pessoa, A. A. Memory Aware Load Balance Strategy on a Parallel Branch-and-bound Application. *Concurrency and Computation: Practice and Experience 27*, 5 (2015), 1122–1144.

[86] Singh, T.; Vara, P. K. Smart Metering the clouds. In *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)* (2009), IEEE, pp. 66–71.

[87] Steffenel, L. A.; Martinasso, M.; Trystram, D. Assessing Contention Effects on MPI_alltoall Communications. In *International Conference on Grid and Pervasive Computing* (2007), Springer, pp. 424–435.

[88] Stolzenberg, R. M. Multiple Regression Analysis. *Handbook of Data Analysis 165* (2004), 208.

[89] Tomić, D.; Car, Z.; Ogrizović, D. Running HPC Applications on Many Million Cores Cloud. In *40th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics* (2017).

[90] Tsuruoka, Y. Cloud Computing-Current Status and Future Directions. *Journal of Information Processing 24*, 2 (2016), 183–194.

[91] Vandekerckhove, J.; Matzke, D.; Wagenmakers, E. Model Comparison and the Principle of Parsimony. *The Oxford Handbook of Computational and Mathematical Psychology* (2014), 300–317.

[92] Xavier, M. G.; Neves, M. V.; Rossi, F. D.; Ferreto, T. C.; Lange, T.; De Rose, C. A. Performance Evaluation of Container-based Virtualization for High Performance Computing Environments. In *21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (2013), IEEE, pp. 233–240.

[93] Xu, C.; Chen, X.; Dick, R. P.; Mao, Z. M. Cache Contention and Application Performance Prediction for Multi-core Systems. In *International Symposium on Performance Analysis of Systems & Software (ISPASS)* (2010), IEEE, pp. 76–86.

[94] Xu, R.; Hugues, M.; Calandra, H.; Chandrasekaran, S.; Chapman, B. Accelerating Kirchhoff Migration on GPU using Directives. In *First Workshop on Accelerator Programming using Directives* (2014), IEEE, pp. 37–46.

[95] Yerneni, S.; Phadke, S.; Bhardwaj, D.; Chakraborty, S.; Rastogi, R. Imaging Subsurface Geology with Seismic Migration on a Computing Cluster. *Current Science 88*, 3 (2005), 468–474.

[96] Yokoyama, D.; Schulze, B.; Kloh, H.; Bandini, M.; Rebello, V. Affinity Aware Scheduling Model of Cluster Nodes in Private Clouds. *Journal of Network and Computer Applications 95* (2017), 94–104.

[97] Younge, A. J.; Henschel, R.; Brown, J. T.; Von Laszewski, G.; Qiu, J.; Fox, G. C. Analysis of Virtualization Technologies for High Performance Computing Environments. In *International Conference on Cloud Computing (CLOUD)* (2011), IEEE, pp. 9–16.

APPENDIX A – Complete Interference Results

Table A.1: Interference results for co-location scheme “A”

Applications		**Real Interference Level**
PTRANS.I1.P6	PTRANS.I1.P6	44.50%
PTRANS.I2.P6	PTRANS.I2.P6	5.31%
DGEMM.I2.P6	DGEMM.I2.P6	7.70%
DGEMM.I1.P6	DGEMM.I1.P6	7.79%
HPL.I2.P6	HPL.I2.P6	11.40%
HPL.I1.P6	HPL.I1.P6	10.75%
PTRANS.I1.P6	PTRANS.I2.P6	22.56%
DGEMM.I2.P6	DGEMM.I1.P6	7.25%
HPL.I2.P6	HPL.I1.P6	10.66%
PTRANS.I1.P6	HPL.I2.P6	18.37%
PTRANS.I1.P6	HPL.I1.P6	18.30%
PTRANS.I1.P6	DGEMM.I2.P6	12.69%
PTRANS.I1.P6	DGEMM.I1.P6	12.71%
HPL.I2.P6	DGEMM.I2.P6	7.45%
HPL.I1.P6	DGEMM.I1.P6	7.46%
HPL.I2.P6	DGEMM.I1.P6	7.91%
HPL.I1.P6	DGEMM.I2.P6	7.80%
PTRANS.I2.P6	HPL.I2.P6	9.96%
PTRANS.I2.P6	HPL.I1.P6	10.47%
PTRANS.I2.P6	DGEMM.I2.P6	9.27%
PTRANS.I2.P6	DGEMM.I1.P6	8.64%
FFT.I2.P4	FFT.I2.P4	20.69%

Table A.1 continued from previous page

Applications		Real Interference Level
FFT.I2.P4	PTRANS.I1.P6	26.39%
FFT.I2.P4	HPL.I2.P6	10.61%
FFT.I2.P4	DGEMM.I2.P6	7.41%
FFT.I2.P4	PTRANS.I2.P6	8.25%
FFT.I2.P4	HPL.I1.P6	11.10%
FFT.I2.P4	DGEMM.I1.P6	6.34%
FFT.I1.P4	FFT.I1.P4	22.76%
FFT.I1.P4	PTRANS.I1.P6	26.51%
FFT.I1.P4	HPL.I2.P6	11.21%
FFT.I1.P4	DGEMM.I2.P6	6.65%
FFT.I1.P4	PTRANS.I2.P6	8.31%
FFT.I1.P4	HPL.I1.P6	11.01%
FFT.I1.P4	DGEMM.I1.P6	6.96%
FFT.I2.P4	FFT.I1.P4	25.29%
MUFITS.I1.P6	MUFITS.I1.P6	10.72%
MUFITS.I1.P6	PTRANS.I1.P6	21.83%
MUFITS.I1.P6	PTRANS.I2.P6	7.32%
MUFITS.I1.P6	HPL.I2.P6	9.83%
MUFITS.I1.P6	HPL.I1.P6	9.74%
MUFITS.I1.P6	DGEMM.I2.P6	5.98%
MUFITS.I1.P6	DGEMM.I1.P6	6.81%
MUFITS.I1.P6	FFT.I2.P4	11.23%
MUFITS.I1.P6	FFT.I1.P4	9.08%
MUFITS.I2.P6	MUFITS.I2.P6	4.02%
MUFITS.I2.P6	PTRANS.I1.P6	23.07%
MUFITS.I2.P6	PTRANS.I2.P6	5.28%
MUFITS.I2.P6	HPL.I2.P6	8.28%
MUFITS.I2.P6	HPL.I1.P6	8.97%
MUFITS.I2.P6	DGEMM.I2.P6	9.14%
MUFITS.I2.P6	DGEMM.I1.P6	7.66%

Table A.1 continued from previous page

Applications		Real Interference Level
MUFITS.I2.P6	FFT.I2.P4	10.07%
MUFITS.I2.P6	FFT.I1.P4	9.79%
MUFITS.I2.P6	MUFITS.I1.P6	8.84%
PKTM.I1.P6	PKTM.I1.P6	0.17%
PKTM.I1.P6	PKTM.I2.P6	0.43%
PKTM.I1.P6	MUFITS.I2.P6	2.08%
PKTM.I1.P6	MUFITS.I1.P6	1.71%
PKTM.I1.P6	HPL.I2.P6	3.93%
PKTM.I1.P6	HPL.I1.P6	4.13%
PKTM.I1.P6	FFT.I2.P4	5.09%
PKTM.I1.P6	FFT.I1.P4	3.81%
PKTM.I1.P6	PTRANS.I1.P6	2.78%
PKTM.I1.P6	PTRANS.I2.P6	0.27%
PKTM.I1.P6	DGEMM.I2.P6	4.07%
PKTM.I1.P6	DGEMM.I1.P6	3.81%
PKTM.I2.P6	MUFITS.I2.P6	1.85%
PKTM.I2.P6	MUFITS.I1.P6	1.35%
PKTM.I2.P6	HPL.I2.P6	2.65%
PKTM.I2.P6	HPL.I1.P6	2.53%
PKTM.I2.P6	FFT.I2.P4	3.66%
PKTM.I2.P6	FFT.I1.P4	4.67%
PKTM.I2.P6	PTRANS.I1.P6	3.01%
PKTM.I2.P6	PTRANS.I2.P6	0.24%
PKTM.I2.P6	DGEMM.I2.P6	3.70%
PKTM.I2.P6	DGEMM.I1.P6	3.28%
PKTM.I2.P6	PKTM.I2.P6	0.03%

Table A.2: Interference results for co-location scheme “B”

Applications			**Real Interference Level**
HPL.I1.P4	PTRANS.I1.P4	DGEMM.I1.P4	11.51%
HPL.I1.P4	HPL.I1.P4	HPL.I1.P4	13.85%
DGEMM.I1.P4	DGEMM.I1.P4	DGEMM.I1.P4	9.60%
PTRANS.I1.P4	PTRANS.I1.P4	PTRANS.I1.P4	37.04%
HPL.I1.P4	HPL.I1.P4	PTRANS.I1.P4	17.45%
HPL.I1.P4	HPL.I1.P4	DGEMM.I1.P4	13.22%
PTRANS.I1.P4	PTRANS.I1.P4	DGEMM.I1.P4	27.03%
PTRANS.I1.P4	PTRANS.I1.P4	HPL.I1.P4	23.28%
DGEMM.I1.P4	DGEMM.I1.P4	HPL.I1.P4	11.15%
DGEMM.I1.P4	DGEMM.I1.P4	PTRANS.I1.P4	12.46%
FFT.I2.P4	FFT.I2.P4	FFT.I2.P4	48.81%
FFT.I2.P4	FFT.I2.P4	HPL.I1.P4	27.82%
FFT.I2.P4	FFT.I2.P4	DGEMM.I1.P4	27.50%
FFT.I2.P4	FFT.I2.P4	PTRANS.I1.P4	42.21%
FFT.I2.P4	HPL.I1.P4	HPL.I1.P4	19.32%
FFT.I2.P4	PTRANS.I1.P4	PTRANS.I1.P4	40.91%
FFT.I2.P4	DGEMM.I1.P4	DGEMM.I1.P4	12.39%
FFT.I2.P4	PTRANS.I1.P4	HPL.I1.P4	24.74%
FFT.I2.P4	PTRANS.I1.P4	DGEMM.I1.P4	20.17%
FFT.I2.P4	HPL.I1.P4	DGEMM.I1.P4	15.62%
MUFITS.I1.P4	PTRANS.I1.P4	PTRANS.I1.P4	24.77%
MUFITS.I1.P4	HPL.I1.P4	HPL.I1.P4	15.20%
MUFITS.I1.P4	DGEMM.I1.P4	DGEMM.I1.P4	10.63%
MUFITS.I1.P4	FFT.I2.P4	FFT.I2.P4	34.83%
MUFITS.I1.P4	MUFITS.I1.P4	PTRANS.I1.P4	19.76%
MUFITS.I1.P4	MUFITS.I1.P4	HPL.I1.P4	20.55%
MUFITS.I1.P4	MUFITS.I1.P4	FFT.I2.P4	25.88%
MUFITS.I1.P4	MUFITS.I1.P4	DGEMM.I1.P4	15.75%
MUFITS.I1.P4	MUFITS.I1.P4	MUFITS.I1.P4	22.85%
MUFITS.I1.P4	PTRANS.I1.P4	DGEMM.I1.P4	14.51%

Table A.2 continued from previous page

Applications			Real Interference Level
MUFITS.I1.P4	PTRANS.I1.P4	HPL.I1.P4	18.90%
MUFITS.I1.P4	DGEMM.I1.P4	HPL.I1.P4	13.20%
MUFITS.I1.P4	PTRANS.I1.P4	FFT.I2.P4	28.76%
MUFITS.I1.P4	DGEMM.I1.P4	FFT.I2.P4	15.82%
MUFITS.I1.P4	FFT.I2.P4	HPL.I1.P4	20.14%
DGEMM.I2.P4	DGEMM.I2.P4	DGEMM.I2.P4	10.54%
DGEMM.I2.P4	DGEMM.I2.P4	HPL.I2.P4	11.66%
DGEMM.I2.P4	DGEMM.I2.P4	PTRANS.I2.P4	6.00%
DGEMM.I2.P4	DGEMM.I2.P4	FFT.I1.P4	11.86%
DGEMM.I2.P4	HPL.I2.P4	HPL.I2.P4	12.77%
DGEMM.I2.P4	HPL.I2.P4	PTRANS.I2.P4	8.67%
DGEMM.I2.P4	HPL.I2.P4	FFT.I1.P4	15.92%
DGEMM.I2.P4	PTRANS.I2.P4	PTRANS.I2.P4	2.08%
DGEMM.I2.P4	PTRANS.I2.P4	FFT.I1.P4	11.72%
DGEMM.I2.P4	FFT.I1.P4	FFT.I1.P4	24.69%
HPL.I2.P4	HPL.I2.P4	HPL.I2.P4	15.39%
HPL.I2.P4	HPL.I2.P4	PTRANS.I2.P4	10.52%
HPL.I2.P4	HPL.I2.P4	FFT.I1.P4	19.75%
HPL.I2.P4	PTRANS.I2.P4	PTRANS.I2.P4	4.99%
HPL.I2.P4	PTRANS.I2.P4	FFT.I1.P4	14.28%
HPL.I2.P4	FFT.I1.P4	FFT.I1.P4	30.96%
PTRANS.I2.P4	PTRANS.I2.P4	PTRANS.I2.P4	0.11%
PTRANS.I2.P4	PTRANS.I2.P4	FFT.I1.P4	14.52%
PTRANS.I2.P4	FFT.I1.P4	FFT.I1.P4	27.72%
FFT.I1.P4	FFT.I1.P4	FFT.I1.P4	47.31%
HPL.I1.P4	HPL.I1.P4	DGEMM.I2.P4	12.94%
HPL.I1.P4	HPL.I1.P4	HPL.I2.P4	15.44%
HPL.I1.P4	HPL.I1.P4	PTRANS.I2.P4	10.62%
HPL.I1.P4	HPL.I1.P4	FFT.I1.P4	19.80%
HPL.I1.P4	DGEMM.I2.P4	DGEMM.I2.P4	11.44%

Table A.2 continued from previous page

Applications			Real Interference Level
HPL.I1.P4	DGEMM.I2.P4	HPL.I2.P4	12.86%
HPL.I1.P4	DGEMM.I2.P4	PTRANS.I2.P4	7.24%
HPL.I1.P4	DGEMM.I2.P4	FFT.I1.P4	15.07%
HPL.I1.P4	HPL.I2.P4	HPL.I2.P4	15.56%
HPL.I1.P4	HPL.I2.P4	PTRANS.I2.P4	10.72%
HPL.I1.P4	HPL.I2.P4	FFT.I1.P4	19.63%
HPL.I1.P4	PTRANS.I2.P4	PTRANS.I2.P4	5.21%
HPL.I1.P4	PTRANS.I2.P4	FFT.I1.P4	16.51%
HPL.I1.P4	FFT.I1.P4	FFT.I1.P4	31.43%
DGEMM.I2.P4	DGEMM.I2.P4	DGEMM.I1.P4	10.15%
DGEMM.I2.P4	DGEMM.I2.P4	PTRANS.I1.P4	15.34%
DGEMM.I2.P4	DGEMM.I2.P4	FFT.I2.P4	21.31%
DGEMM.I2.P4	HPL.I2.P4	PTRANS.I1.P4	16.89%
DGEMM.I2.P4	HPL.I2.P4	FFT.I2.P4	15.63%
DGEMM.I2.P4	HPL.I2.P4	DGEMM.I1.P4	11.72%
DGEMM.I2.P4	PTRANS.I2.P4	DGEMM.I1.P4	8.22%
DGEMM.I2.P4	PTRANS.I2.P4	PTRANS.I1.P4	12.42%
DGEMM.I2.P4	PTRANS.I2.P4	FFT.I2.P4	11.45%
DGEMM.I2.P4	FFT.I1.P4	DGEMM.I1.P4	12.70%
DGEMM.I2.P4	FFT.I1.P4	PTRANS.I1.P4	22.68%
DGEMM.I2.P4	FFT.I1.P4	FFT.I2.P4	23.52%
DGEMM.I2.P4	DGEMM.I1.P4	DGEMM.I1.P4	9.77%
DGEMM.I2.P4	DGEMM.I1.P4	HPL.I1.P4	11.06%
DGEMM.I2.P4	DGEMM.I1.P4	PTRANS.I1.P4	14.51%
DGEMM.I2.P4	DGEMM.I1.P4	FFT.I2.P4	13.27%
DGEMM.I2.P4	HPL.I1.P4	PTRANS.I1.P4	15.95%
DGEMM.I2.P4	HPL.I1.P4	FFT.I2.P4	14.48%
DGEMM.I2.P4	PTRANS.I1.P4	PTRANS.I1.P4	23.22%
DGEMM.I2.P4	PTRANS.I1.P4	FFT.I2.P4	20.10%
DGEMM.I2.P4	FFT.I2.P4	FFT.I2.P4	25.75%

Table A.2 continued from previous page

Applications			Real Interference Level
HPL.I2.P4	HPL.I2.P4	DGEMM.I1.P4	13.39%
HPL.I2.P4	HPL.I2.P4	PTRANS.I1.P4	19.73%
HPL.I2.P4	HPL.I2.P4	FFT.I2.P4	19.44%
HPL.I2.P4	PTRANS.I2.P4	DGEMM.I1.P4	9.12%
HPL.I2.P4	PTRANS.I2.P4	PTRANS.I1.P4	15.42%
HPL.I2.P4	PTRANS.I2.P4	FFT.I2.P4	14.50%
HPL.I2.P4	FFT.I1.P4	DGEMM.I1.P4	15.59%
HPL.I2.P4	FFT.I1.P4	PTRANS.I1.P4	31.83%
HPL.I2.P4	FFT.I1.P4	FFT.I2.P4	29.78%
HPL.I2.P4	DGEMM.I1.P4	DGEMM.I1.P4	10.89%
HPL.I2.P4	DGEMM.I1.P4	HPL.I1.P4	12.97%
HPL.I2.P4	DGEMM.I1.P4	PTRANS.I1.P4	15.84%
HPL.I2.P4	DGEMM.I1.P4	FFT.I2.P4	15.29%
HPL.I2.P4	HPL.I1.P4	PTRANS.I1.P4	19.52%
HPL.I2.P4	HPL.I1.P4	FFT.I2.P4	19.32%
HPL.I2.P4	PTRANS.I1.P4	PTRANS.I1.P4	30.32%
HPL.I2.P4	PTRANS.I1.P4	FFT.I2.P4	27.74%
HPL.I2.P4	FFT.I2.P4	FFT.I2.P4	29.95%
PTRANS.I2.P4	PTRANS.I2.P4	DGEMM.I1.P4	3.73%
PTRANS.I2.P4	PTRANS.I2.P4	PTRANS.I1.P4	8.30%
PTRANS.I2.P4	PTRANS.I2.P4	FFT.I2.P4	11.57%
PTRANS.I2.P4	FFT.I1.P4	DGEMM.I1.P4	11.67%
PTRANS.I2.P4	FFT.I1.P4	PTRANS.I1.P4	21.62%
PTRANS.I2.P4	FFT.I1.P4	FFT.I2.P4	24.85%
PTRANS.I2.P4	DGEMM.I1.P4	DGEMM.I1.P4	7.71%
PTRANS.I2.P4	DGEMM.I1.P4	HPL.I1.P4	9.02%
PTRANS.I2.P4	DGEMM.I1.P4	PTRANS.I1.P4	12.18%
PTRANS.I2.P4	DGEMM.I1.P4	FFT.I2.P4	11.80%
PTRANS.I2.P4	HPL.I1.P4	PTRANS.I1.P4	15.63%
PTRANS.I2.P4	HPL.I1.P4	FFT.I2.P4	14.40%

Table A.2 continued from previous page

Applications			Real Interference Level
PTRANS.I2.P4	PTRANS.I1.P4	PTRANS.I1.P4	24.20%
PTRANS.I2.P4	PTRANS.I1.P4	FFT.I2.P4	24.60%
PTRANS.I2.P4	FFT.I2.P4	FFT.I2.P4	32.47%
FFT.I1.P4	DGEMM.I1.P4	DGEMM.I1.P4	11.42%
FFT.I1.P4	DGEMM.I1.P4	HPL.I1.P4	15.17%
FFT.I1.P4	DGEMM.I1.P4	PTRANS.I1.P4	22.63%
FFT.I1.P4	DGEMM.I1.P4	FFT.I2.P4	23.91%
FFT.I1.P4	HPL.I1.P4	PTRANS.I1.P4	27.54%
FFT.I1.P4	HPL.I1.P4	FFT.I2.P4	27.09%
FFT.I1.P4	PTRANS.I1.P4	PTRANS.I1.P4	42.64%
FFT.I1.P4	PTRANS.I1.P4	FFT.I2.P4	41.31%
FFT.I1.P4	FFT.I2.P4	FFT.I2.P4	47.46%
FFT.I1.P4	FFT.I1.P4	DGEMM.I1.P4	26.35%
FFT.I1.P4	FFT.I1.P4	PTRANS.I1.P4	40.06%
FFT.I1.P4	FFT.I1.P4	FFT.I2.P4	45.26%
DGEMM.I2.P4	DGEMM.I2.P4	MUFITS.I2.P4	9.07%
DGEMM.I2.P4	DGEMM.I2.P4	MUFITS.I1.P4	12.47%
DGEMM.I2.P4	HPL.I2.P4	MUFITS.I2.P4	9.44%
DGEMM.I2.P4	HPL.I2.P4	MUFITS.I1.P4	11.99%
DGEMM.I2.P4	PTRANS.I2.P4	MUFITS.I2.P4	3.26%
DGEMM.I2.P4	PTRANS.I2.P4	MUFITS.I1.P4	10.98%
DGEMM.I2.P4	FFT.I1.P4	MUFITS.I2.P4	12.55%
DGEMM.I2.P4	FFT.I1.P4	MUFITS.I1.P4	16.73%
DGEMM.I2.P4	DGEMM.I1.P4	MUFITS.I2.P4	8.99%
DGEMM.I2.P4	DGEMM.I1.P4	MUFITS.I1.P4	10.10%
DGEMM.I2.P4	HPL.I1.P4	MUFITS.I2.P4	9.84%
DGEMM.I2.P4	HPL.I1.P4	MUFITS.I1.P4	12.50%
DGEMM.I2.P4	PTRANS.I1.P4	MUFITS.I2.P4	13.19%
DGEMM.I2.P4	PTRANS.I1.P4	MUFITS.I1.P4	18.55%
DGEMM.I2.P4	FFT.I2.P4	MUFITS.I2.P4	13.61%

Table A.2 continued from previous page

Applications			Real Interference Level
DGEMM.I2.P4	FFT.I2.P4	MUFITS.I1.P4	17.51%
DGEMM.I2.P4	MUFITS.I2.P4	MUFITS.I2.P4	7.21%
DGEMM.I2.P4	MUFITS.I2.P4	MUFITS.I1.P4	11.84%
DGEMM.I2.P4	MUFITS.I1.P4	MUFITS.I1.P4	14.09%
HPL.I2.P4	HPL.I2.P4	MUFITS.I2.P4	12.23%
HPL.I2.P4	HPL.I2.P4	MUFITS.I1.P4	16.15%
HPL.I2.P4	PTRANS.I2.P4	MUFITS.I2.P4	6.17%
HPL.I2.P4	PTRANS.I2.P4	MUFITS.I1.P4	12.48%
HPL.I2.P4	FFT.I1.P4	MUFITS.I2.P4	16.08%
HPL.I2.P4	FFT.I1.P4	MUFITS.I1.P4	23.60%
HPL.I2.P4	DGEMM.I1.P4	MUFITS.I2.P4	10.33%
HPL.I2.P4	DGEMM.I1.P4	MUFITS.I1.P4	12.82%
HPL.I2.P4	HPL.I1.P4	MUFITS.I2.P4	11.66%
HPL.I2.P4	HPL.I1.P4	MUFITS.I1.P4	15.72%
HPL.I2.P4	PTRANS.I1.P4	MUFITS.I2.P4	16.66%
HPL.I2.P4	PTRANS.I1.P4	MUFITS.I1.P4	22.71%
HPL.I2.P4	FFT.I2.P4	MUFITS.I2.P4	20.05%
HPL.I2.P4	FFT.I2.P4	MUFITS.I1.P4	22.43%
HPL.I2.P4	MUFITS.I2.P4	MUFITS.I2.P4	8.76%
HPL.I2.P4	MUFITS.I2.P4	MUFITS.I1.P4	13.60%
HPL.I2.P4	MUFITS.I1.P4	MUFITS.I1.P4	17.91%
PTRANS.I2.P4	PTRANS.I2.P4	MUFITS.I2.P4	0.85%
PTRANS.I2.P4	PTRANS.I2.P4	MUFITS.I1.P4	9.03%
PTRANS.I2.P4	FFT.I1.P4	MUFITS.I2.P4	10.39%
PTRANS.I2.P4	FFT.I1.P4	MUFITS.I1.P4	18.12%
PTRANS.I2.P4	DGEMM.I1.P4	MUFITS.I2.P4	5.36%
PTRANS.I2.P4	DGEMM.I1.P4	MUFITS.I1.P4	10.47%
PTRANS.I2.P4	HPL.I1.P4	MUFITS.I2.P4	6.56%
PTRANS.I2.P4	HPL.I1.P4	MUFITS.I1.P4	12.29%
PTRANS.I2.P4	PTRANS.I1.P4	MUFITS.I2.P4	11.38%

Table A.2 continued from previous page

Applications			Real Interference Level
PTRANS.I2.P4	PTRANS.I1.P4	MUFITS.I1.P4	19.10%
PTRANS.I2.P4	FFT.I2.P4	MUFITS.I2.P4	11.84%
PTRANS.I2.P4	FFT.I2.P4	MUFITS.I1.P4	19.12%
PTRANS.I2.P4	MUFITS.I2.P4	MUFITS.I2.P4	3.39%
PTRANS.I2.P4	MUFITS.I2.P4	MUFITS.I1.P4	10.31%
PTRANS.I2.P4	MUFITS.I1.P4	MUFITS.I1.P4	16.10%
FFT.I1.P4	FFT.I1.P4	MUFITS.I2.P4	28.20%
FFT.I1.P4	FFT.I1.P4	MUFITS.I1.P4	35.12%
FFT.I1.P4	DGEMM.I1.P4	MUFITS.I2.P4	12.04%
FFT.I1.P4	DGEMM.I1.P4	MUFITS.I1.P4	18.17%
FFT.I1.P4	HPL.I1.P4	MUFITS.I2.P4	16.16%
FFT.I1.P4	HPL.I1.P4	MUFITS.I1.P4	22.95%
FFT.I1.P4	PTRANS.I1.P4	MUFITS.I2.P4	25.08%
FFT.I1.P4	PTRANS.I1.P4	MUFITS.I1.P4	34.24%
FFT.I1.P4	FFT.I2.P4	MUFITS.I2.P4	25.71%
FFT.I1.P4	FFT.I2.P4	MUFITS.I1.P4	34.57%
FFT.I1.P4	MUFITS.I2.P4	MUFITS.I2.P4	13.82%
FFT.I1.P4	MUFITS.I2.P4	MUFITS.I1.P4	19.81%
FFT.I1.P4	MUFITS.I1.P4	MUFITS.I1.P4	28.69%
DGEMM.I1.P4	DGEMM.I1.P4	MUFITS.I2.P4	8.48%
DGEMM.I1.P4	HPL.I1.P4	MUFITS.I2.P4	10.24%
DGEMM.I1.P4	PTRANS.I1.P4	MUFITS.I2.P4	13.40%
DGEMM.I1.P4	FFT.I2.P4	MUFITS.I2.P4	15.25%
DGEMM.I1.P4	MUFITS.I2.P4	MUFITS.I2.P4	7.48%
DGEMM.I1.P4	MUFITS.I2.P4	MUFITS.I1.P4	10.87%
HPL.I1.P4	HPL.I1.P4	MUFITS.I2.P4	12.04%
HPL.I1.P4	PTRANS.I1.P4	MUFITS.I2.P4	15.76%
HPL.I1.P4	FFT.I2.P4	MUFITS.I2.P4	16.18%
HPL.I1.P4	MUFITS.I2.P4	MUFITS.I2.P4	8.98%
HPL.I1.P4	MUFITS.I2.P4	MUFITS.I1.P4	12.94%

Table A.2 continued from previous page

Applications			Real Interference Level
PTRANS.I1.P4	PTRANS.I1.P4	MUFITS.I2.P4	25.44%
PTRANS.I1.P4	FFT.I2.P4	MUFITS.I2.P4	23.36%
PTRANS.I1.P4	MUFITS.I2.P4	MUFITS.I2.P4	13.26%
PTRANS.I1.P4	MUFITS.I2.P4	MUFITS.I1.P4	20.41%
FFT.I2.P4	FFT.I2.P4	MUFITS.I2.P4	31.09%
FFT.I2.P4	MUFITS.I2.P4	MUFITS.I2.P4	14.02%
FFT.I2.P4	MUFITS.I2.P4	MUFITS.I1.P4	21.81%
MUFITS.I2.P4	MUFITS.I2.P4	MUFITS.I2.P4	4.03%
MUFITS.I2.P4	MUFITS.I2.P4	MUFITS.I1.P4	10.38%
MUFITS.I2.P4	MUFITS.I1.P4	MUFITS.I1.P4	15.90%
PKTM.I1.P4	PKTM.I1.P4	PKTM.I1.P4	3.04%
PKTM.I1.P4	PKTM.I1.P4	HPL.I1.P4	6.71%
PKTM.I1.P4	PKTM.I1.P4	HPL.I2.P4	6.40%
PKTM.I1.P4	PKTM.I1.P4	DGEMM.I2.P4	6.05%
PKTM.I1.P4	PKTM.I1.P4	DGEMM.I1.P4	5.99%
PKTM.I1.P4	PKTM.I1.P4	FFT.I2.P4	7.41%
PKTM.I1.P4	PKTM.I1.P4	FFT.I1.P4	5.97%
PKTM.I1.P4	PKTM.I1.P4	PTRANS.I1.P4	9.06%
PKTM.I1.P4	PKTM.I1.P4	PTRANS.I2.P4	5.21%
PKTM.I1.P4	HPL.I1.P4	HPL.I1.P4	13.95%
PKTM.I1.P4	HPL.I1.P4	HPL.I2.P4	10.65%
PKTM.I1.P4	HPL.I1.P4	DGEMM.I2.P4	9.00%
PKTM.I1.P4	HPL.I1.P4	DGEMM.I1.P4	9.40%
PKTM.I1.P4	HPL.I1.P4	FFT.I2.P4	12.48%
PKTM.I1.P4	HPL.I1.P4	FFT.I1.P4	10.20%
PKTM.I1.P4	HPL.I1.P4	PTRANS.I1.P4	12.86%
PKTM.I1.P4	HPL.I1.P4	PTRANS.I2.P4	8.53%
PKTM.I1.P4	HPL.I2.P4	HPL.I2.P4	10.02%
PKTM.I1.P4	HPL.I2.P4	DGEMM.I2.P4	8.60%
PKTM.I1.P4	HPL.I2.P4	DGEMM.I1.P4	8.55%

Table A.2 continued from previous page

Applications			Real Interference Level
PKTM.I1.P4	HPL.I2.P4	FFT.I2.P4	12.08%
PKTM.I1.P4	HPL.I2.P4	FFT.I1.P4	9.80%
PKTM.I1.P4	HPL.I2.P4	PTRANS.I1.P4	13.05%
PKTM.I1.P4	HPL.I2.P4	PTRANS.I2.P4	6.33%
PKTM.I1.P4	DGEMM.I2.P4	DGEMM.I2.P4	7.76%
PKTM.I1.P4	DGEMM.I2.P4	DGEMM.I1.P4	7.70%
PKTM.I1.P4	DGEMM.I2.P4	FFT.I2.P4	10.82%
PKTM.I1.P4	DGEMM.I2.P4	FFT.I1.P4	7.67%
PKTM.I1.P4	DGEMM.I2.P4	PTRANS.I1.P4	10.97%
PKTM.I1.P4	DGEMM.I2.P4	PTRANS.I2.P4	7.67%
PKTM.I1.P4	DGEMM.I1.P4	DGEMM.I1.P4	8.54%
PKTM.I1.P4	DGEMM.I1.P4	FFT.I2.P4	10.31%
PKTM.I1.P4	DGEMM.I1.P4	FFT.I1.P4	8.45%
PKTM.I1.P4	DGEMM.I1.P4	PTRANS.I1.P4	12.14%
PKTM.I1.P4	DGEMM.I1.P4	PTRANS.I2.P4	5.53%
PKTM.I1.P4	FFT.I2.P4	FFT.I2.P4	15.15%
PKTM.I1.P4	FFT.I2.P4	FFT.I1.P4	18.40%
PKTM.I1.P4	FFT.I2.P4	PTRANS.I1.P4	21.80%
PKTM.I1.P4	FFT.I2.P4	PTRANS.I2.P4	14.51%
PKTM.I1.P4	FFT.I1.P4	FFT.I1.P4	15.29%
PKTM.I1.P4	FFT.I1.P4	PTRANS.I1.P4	15.98%
PKTM.I1.P4	FFT.I1.P4	PTRANS.I2.P4	8.82%
PKTM.I1.P4	PTRANS.I1.P4	PTRANS.I1.P4	19.23%
PKTM.I1.P4	PTRANS.I1.P4	PTRANS.I2.P4	9.93%
PKTM.I1.P4	PTRANS.I2.P4	PTRANS.I2.P4	2.12%
PKTM.I2.P4	PKTM.I2.P4	PKTM.I2.P4	5.05%
PKTM.I2.P4	PKTM.I2.P4	HPL.I1.P4	7.85%
PKTM.I2.P4	PKTM.I2.P4	HPL.I2.P4	7.46%
PKTM.I2.P4	PKTM.I2.P4	DGEMM.I2.P4	6.89%
PKTM.I2.P4	PKTM.I2.P4	DGEMM.I1.P4	6.83%

Table A.2 continued from previous page

Applications			Real Interference Level
PKTM.I2.P4	PKTM.I2.P4	FFT.I2.P4	7.21%
PKTM.I2.P4	PKTM.I2.P4	FFT.I1.P4	5.24%
PKTM.I2.P4	PKTM.I2.P4	PTRANS.I1.P4	8.37%
PKTM.I2.P4	PKTM.I2.P4	PTRANS.I2.P4	3.64%
PKTM.I2.P4	HPL.I1.P4	HPL.I1.P4	10.66%
PKTM.I2.P4	HPL.I1.P4	HPL.I2.P4	10.09%
PKTM.I2.P4	HPL.I1.P4	DGEMM.I2.P4	8.95%
PKTM.I2.P4	HPL.I1.P4	DGEMM.I1.P4	9.06%
PKTM.I2.P4	HPL.I1.P4	FFT.I2.P4	12.62%
PKTM.I2.P4	HPL.I1.P4	FFT.I1.P4	10.34%
PKTM.I2.P4	HPL.I1.P4	PTRANS.I1.P4	13.46%
PKTM.I2.P4	HPL.I1.P4	PTRANS.I2.P4	9.11%
PKTM.I2.P4	HPL.I2.P4	HPL.I2.P4	10.86%
PKTM.I2.P4	HPL.I2.P4	DGEMM.I2.P4	9.18%
PKTM.I2.P4	HPL.I2.P4	DGEMM.I1.P4	8.78%
PKTM.I2.P4	HPL.I2.P4	FFT.I2.P4	10.98%
PKTM.I2.P4	HPL.I2.P4	FFT.I1.P4	10.90%
PKTM.I2.P4	HPL.I2.P4	PTRANS.I1.P4	12.59%
PKTM.I2.P4	HPL.I2.P4	PTRANS.I2.P4	8.84%
PKTM.I2.P4	DGEMM.I2.P4	DGEMM.I2.P4	10.62%
PKTM.I2.P4	DGEMM.I2.P4	DGEMM.I1.P4	7.94%
PKTM.I2.P4	DGEMM.I2.P4	FFT.I2.P4	8.06%
PKTM.I2.P4	DGEMM.I2.P4	FFT.I1.P4	8.66%
PKTM.I2.P4	DGEMM.I2.P4	PTRANS.I1.P4	11.19%
PKTM.I2.P4	DGEMM.I2.P4	PTRANS.I2.P4	7.94%
PKTM.I2.P4	DGEMM.I1.P4	DGEMM.I1.P4	8.33%
PKTM.I2.P4	DGEMM.I1.P4	FFT.I2.P4	9.97%
PKTM.I2.P4	DGEMM.I1.P4	FFT.I1.P4	9.50%
PKTM.I2.P4	DGEMM.I1.P4	PTRANS.I1.P4	13.90%
PKTM.I2.P4	DGEMM.I1.P4	PTRANS.I2.P4	8.04%

Table A.2 continued from previous page

Applications			Real Interference Level
PKTM.I2.P4	FFT.I2.P4	FFT.I2.P4	14.66%
PKTM.I2.P4	FFT.I2.P4	FFT.I1.P4	11.70%
PKTM.I2.P4	FFT.I2.P4	PTRANS.I1.P4	16.80%
PKTM.I2.P4	FFT.I2.P4	PTRANS.I2.P4	9.88%
PKTM.I2.P4	FFT.I1.P4	FFT.I1.P4	14.71%
PKTM.I2.P4	FFT.I1.P4	PTRANS.I1.P4	16.35%
PKTM.I2.P4	FFT.I1.P4	PTRANS.I2.P4	9.11%
PKTM.I2.P4	PTRANS.I1.P4	PTRANS.I1.P4	18.71%
PKTM.I2.P4	PTRANS.I1.P4	PTRANS.I2.P4	12.37%
PKTM.I2.P4	PTRANS.I2.P4	PTRANS.I2.P4	2.07%
PKTM.I2.P4	PKTM.I1.P4	PKTM.I2.P4	4.49%
PKTM.I2.P4	PKTM.I1.P4	HPL.I1.P4	7.29%
PKTM.I2.P4	PKTM.I1.P4	HPL.I2.P4	6.70%
PKTM.I2.P4	PKTM.I1.P4	DGEMM.I2.P4	6.29%
PKTM.I2.P4	PKTM.I1.P4	DGEMM.I1.P4	8.21%
PKTM.I2.P4	PKTM.I1.P4	FFT.I2.P4	5.90%
PKTM.I2.P4	PKTM.I1.P4	FFT.I1.P4	4.45%
PKTM.I2.P4	PKTM.I1.P4	PTRANS.I1.P4	8.08%
PKTM.I2.P4	PKTM.I1.P4	PTRANS.I2.P4	2.92%
PKTM.I1.P4	PKTM.I1.P4	PKTM.I2.P4	3.76%
PKTM.I1.P4	PKTM.I1.P4	MUFITS.I2.P4	3.21%
PKTM.I1.P4	MUFITS.I2.P4	HPL.I1.P4	7.49%
PKTM.I1.P4	MUFITS.I2.P4	HPL.I2.P4	6.59%
PKTM.I1.P4	MUFITS.I2.P4	DGEMM.I2.P4	6.77%
PKTM.I1.P4	MUFITS.I2.P4	DGEMM.I1.P4	6.29%
PKTM.I1.P4	MUFITS.I2.P4	FFT.I2.P4	10.84%
PKTM.I1.P4	MUFITS.I2.P4	FFT.I1.P4	6.83%
PKTM.I1.P4	MUFITS.I2.P4	PTRANS.I1.P4	9.50%
PKTM.I1.P4	MUFITS.I2.P4	PTRANS.I2.P4	3.01%
PKTM.I2.P4	PKTM.I2.P4	MUFITS.I2.P4	4.50%

Table A.2 continued from previous page

Applications			Real Interference Level
PKTM.I2.P4	MUFITS.I2.P4	HPL.I1.P4	7.55%
PKTM.I2.P4	MUFITS.I2.P4	HPL.I2.P4	7.46%
PKTM.I2.P4	MUFITS.I2.P4	DGEMM.I2.P4	6.41%
PKTM.I2.P4	MUFITS.I2.P4	DGEMM.I1.P4	7.61%
PKTM.I2.P4	MUFITS.I2.P4	FFT.I2.P4	9.33%
PKTM.I2.P4	MUFITS.I2.P4	FFT.I1.P4	6.86%
PKTM.I2.P4	MUFITS.I2.P4	PTRANS.I1.P4	11.15%
PKTM.I2.P4	MUFITS.I2.P4	PTRANS.I2.P4	3.57%
PKTM.I1.P4	PKTM.I1.P4	MUFITS.I1.P4	3.88%
PKTM.I1.P4	MUFITS.I1.P4	HPL.I1.P4	8.87%
PKTM.I1.P4	MUFITS.I1.P4	HPL.I2.P4	8.62%
PKTM.I1.P4	MUFITS.I1.P4	DGEMM.I2.P4	7.55%
PKTM.I1.P4	MUFITS.I1.P4	DGEMM.I1.P4	7.06%
PKTM.I1.P4	MUFITS.I1.P4	FFT.I2.P4	11.30%
PKTM.I1.P4	MUFITS.I1.P4	FFT.I1.P4	11.99%
PKTM.I1.P4	MUFITS.I1.P4	PTRANS.I1.P4	15.02%
PKTM.I1.P4	MUFITS.I1.P4	PTRANS.I2.P4	7.38%
PKTM.I2.P4	PKTM.I2.P4	MUFITS.I1.P4	4.26%
PKTM.I2.P4	MUFITS.I1.P4	HPL.I1.P4	9.03%
PKTM.I2.P4	MUFITS.I1.P4	HPL.I2.P4	8.67%
PKTM.I2.P4	MUFITS.I1.P4	DGEMM.I2.P4	6.27%
PKTM.I2.P4	MUFITS.I1.P4	DGEMM.I1.P4	7.21%
PKTM.I2.P4	MUFITS.I1.P4	FFT.I2.P4	13.09%
PKTM.I2.P4	MUFITS.I1.P4	FFT.I1.P4	10.81%
PKTM.I2.P4	MUFITS.I1.P4	PTRANS.I1.P4	13.29%
PKTM.I2.P4	MUFITS.I1.P4	PTRANS.I2.P4	5.60%
PKTM.I1.P4	MUFITS.I2.P4	MUFITS.I2.P4	3.39%
PKTM.I1.P4	MUFITS.I1.P4	MUFITS.I1.P4	12.60%
PKTM.I1.P4	MUFITS.I2.P4	MUFITS.I1.P4	5.29%
PKTM.I2.P4	MUFITS.I2.P4	MUFITS.I2.P4	3.78%

Table A.2 continued from previous page

Applications			Real Interference Level
PKTM.I2.P4	MUFITS.I1.P4	MUFITS.I1.P4	10.51%
PKTM.I2.P4	MUFITS.I2.P4	MUFITS.I1.P4	6.70%
PKTM.I2.P4	PKTM.I1.P4	MUFITS.I2.P4	2.96%
PKTM.I2.P4	PKTM.I1.P4	MUFITS.I1.P4	3.47%

Table A.3: Interference results for co-location scheme "C"

Applications						Real Interference Level
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	46.25%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	33.98%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	42.72%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	31.09%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	MUFITS.I1.P2	35.81%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	27.17%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	33.75%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	24.75%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	MUFITS.I1.P2	28.43%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	43.57%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	33.26%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	MUFITS.I1.P2	36.73%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	23.59%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	26.54%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	38.01%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	24.17%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	29.55%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	22.64%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	25.24%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	38.05%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	26.90%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	31.48%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	20.59%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	22.72%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	26.60%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	48.32%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	35.57%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	41.47%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	25.97%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	29.69%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	33.14%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	19.48%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	21.52%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	25.20%
PTRANS.I1.P2	PTRANS.I1.P2	PTRANS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	29.94%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	20.28%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	24.65%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	19.67%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	20.65%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	30.13%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	22.53%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	25.00%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	18.11%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	19.78%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	22.39%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	37.38%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	27.79%

Table A.3 continued from previous page

Applications						Real Interference Level
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	32.93%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	21.14%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	22.93%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	26.55%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	16.62%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	18.06%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.00%
PTRANS.I1.P2	PTRANS.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	23.34%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	46.06%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	34.76%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	39.55%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	27.76%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	30.20%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	35.61%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	22.51%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	23.97%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	27.69%
PTRANS.I1.P2	PTRANS.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	34.13%
PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	15.82%
PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	17.48%
PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	19.27%
PTRANS.I1.P2	PTRANS.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	21.20%
PTRANS.I1.P2	PTRANS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	25.51%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	18.81%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	22.39%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	17.59%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	19.38%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	26.39%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	20.02%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	22.02%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	16.43%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	17.66%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	19.50%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	30.53%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	23.28%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	27.19%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	18.69%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	20.67%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	22.87%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	15.72%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	16.19%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.58%
PTRANS.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.51%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	39.99%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	28.97%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	34.69%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	23.71%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	25.37%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	28.51%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	19.26%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	19.82%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	21.52%
PTRANS.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	24.74%
PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	14.90%
PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	15.16%
PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	16.43%
PTRANS.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	18.53%
PTRANS.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.29%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	50.15%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	37.84%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	42.22%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	28.98%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	32.37%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	36.60%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	21.22%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	24.43%

Table A.3 continued from previous page

Applications						Real Interference Level
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	27.03%
PTRANS.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	31.88%
PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	16.27%
PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	17.42%
PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.24%
PTRANS.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	25.36%
PTRANS.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	27.30%
PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	14.13%
PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	14.56%
PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	15.50%
PTRANS.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.27%
PTRANS.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.10%
PTRANS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	25.30%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	17.70%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	19.25%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	15.80%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	17.64%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	22.78%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	19.14%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	20.67%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	16.44%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	16.49%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.74%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	27.37%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	21.53%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	23.64%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	17.75%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	18.27%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	21.58%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	15.87%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	15.92%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	18.08%
HPL.I1.P2	HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	18.87%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	35.45%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	26.41%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	29.06%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	20.27%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	22.05%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	27.63%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	19.39%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	20.22%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	21.56%
HPL.I1.P2	HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	25.00%
HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	15.12%
HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	15.07%
HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	16.02%
HPL.I1.P2	HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.85%
HPL.I1.P2	HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.04%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	42.98%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	31.81%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	34.86%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	21.97%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	26.75%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	33.04%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	18.71%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	20.71%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	22.70%
HPL.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	26.33%
HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	15.46%
HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	16.74%
HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	18.24%
HPL.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.54%
HPL.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	24.36%
HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	14.45%
HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	14.73%
HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	14.83%

Table A.3 continued from previous page

Applications						Real Interference Level
HPL.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	15.50%
HPL.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.88%
HPL.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	20.70%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	59.63%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	40.05%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	46.36%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	30.21%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	34.50%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	39.21%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	23.17%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	25.06%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	29.42%
FFT.I1.P2	FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	36.02%
FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	19.37%
FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	20.29%
FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	21.75%
FFT.I1.P2	FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	24.97%
FFT.I1.P2	FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	30.73%
FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	15.40%
FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	15.60%
FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.02%
FFT.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	19.46%
FFT.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	23.00%
FFT.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	26.37%
DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	14.22%
DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	13.98%
DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	14.24%
DGEMM.I1.P2	DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	16.71%
DGEMM.I1.P2	DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	17.51%
DGEMM.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	19.94%
MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	MUFITS.I1.P2	22.95%

Table A.4: Interference results for co-location scheme “D”

Applications												Real Interference Level
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	77.04%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	65.89%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	56.74%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	52.36%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	40.55%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	32.85%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	27.81%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	24.87%
PT.I3.P1	PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	24.60%
PT.I3.P1	PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	21.23%
PT.I3.P1	PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	16.68%
PT.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	13.70%
DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	DG.I3.P1	14.31%

Due to space constraint, we shorten applications names in Table A.4. So, “PT.I3.P1” and “DG.I3.P1” means “PTRANS.I3.P1” and “DGEMM.I3.P1”, respectively.

APPENDIX B – Published Papers

- Alves, M.; Drummond, L. Análise de Desempenho de um Simulador de Reservatórios de Petróleo em um Ambiente de Computação em Nuvem. In *XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD) (2014).*

- Alves, M.; Pestana, R.; Drummond, L. Accelerating Pre-stack Kirchhoff Time Migration by using SIMD Vector Instructions. *In XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD) (2015).*
  Best paper award.

- Alves, M.; Drummond, L. Um Modelo Quantitativo para Predição de Interferência entre Aplicações de Alto Desempenho em Ambientes Virtualizados. In *II Escola Regional de Alto Desempenho do Rio de Janeiro (ERAD-RJ) (2016).*

- Alves, M.; Pestana, R.; Silva, R.; Drummond, L. Accelerating pre-stack Kirchhoff Time Migration by Manual Vectorization. *Concurrency and Computation: Practice and Experience (2016).*

- Alves, M.; Drummond, L. A Multivariate and Quantitative Model for Predicting Cross-application Interference in Virtual Environments. *Journal of Systems and Software (2017).*