A Domain Specific Approach to High Performance Heterogeneous Computing

Gordon Inggs, *Student Member, IEEE,* David B. Thomas, *Member, IEEE,* and Wayne Luk, *Fellow, IEEE*

Abstract—Users of heterogeneous computing systems face two problems: firstly, in understanding the trade-off relationships between the observable characteristics of their applications, such as latency and quality of the result, and secondly, how to exploit knowledge of these characteristics to allocate work to distributed computing platforms efficiently. A domain specific approach addresses both of these problems. By considering a subset of operations or functions, models of the observable characteristics or domain metrics may be formulated in advance, and populated at run-time for task instances. These metric models can then be used to express the allocation of work as a constrained integer program.
These claims are illustrated using the example domain of derivatives pricing in computational finance, with the domain metrics of workload latency and pricing accuracy. For a large, varied workload of 128 Black-Scholes and Heston model-based option pricing tasks, running upon a diverse array of 16 Multicore CPUs, GPUs and FPGAs platforms, predictions made by models of both the makespan and accuracy are generally within 10% of the run-time performance. When these models are used as inputs to machine learning and MILP-based workload allocation approaches, a latency improvement of up to 24 and 270 times over the heuristic approach is seen.

1 Introduction

The following vignette illustrates the research problem that we address in this paper:

Julia is a financial analyst at the Bank of England that monitors counterparty risk between investment banks. She is highly qualified in statistics and financial economics, and relies heavily on computational finance techniques to evaluate the derivative contracts that exist between investment banks. However, beyond the specialised programming environment that she uses, she knows little about computing and often runs her calculations for days on her laptop.

She learns that a cluster of heterogeneous computing systems could massively accelerate her computations. She manages to cobble one together using the Bank's spare servers and cloud-based resources. Through the use of an open source application framework, she is soon able to execute her problems upon all of the heterogeneous computing platforms. However, she has no idea about how long a problem is going to take on a given platform. Furthermore, she is also mystified as to the relationship between the statistical accuracy she requires and the time it takes to evaluate her problems. Unable to understand the relationships between the metrics she cares about, she finds that some workloads take even longer on the cluster than on her laptop's CPU!

Julia clearly needs a tool to help her not only understand the resources at her disposal, but also how to use them efficiently.

• G.E. Inggs and D.B. Thomas are with the Circuits and Systems Group in the Department of Electrical and Electronic Engineering at Imperial College London
E-mail: gordon.e.inggs@ieee.org
• W. Luk is with the Custom Computing Group in Department of Computing at Imperial College London

1.1 Problem Statement

Julia might be a fiction, but the problems she faces are a reality for the increasing number of high performance computing application programmers. They have two problems:

1) Understanding the relationships between the run-time characteristics of their application tasks on heterogeneous computing platforms.
2) Allocating tasks to the available platforms so as optimise these run-time characteristics.

In this paper, we both describe and demonstrate practically an approach to high performance, heterogeneous computing that addresses these problems. Our approach is premised on only supporting a subset of operations across all heterogeneous platforms. Computational application domains provide a natural means to limit the operations supported without overly inhibiting programmers, and hence our approach is a domain specific one.

We use the empirical definition of application domains as used in programming research [1], [2], [3], i.e. an identifiable category of computational activities where a small number of computational operations account for all or a disproportionately high proportion of the computations performed. For example, within the domain of Linear Algebra, vector arithmetic is used disproportionately more often than other operations. Hence, by focusing on supporting these frequently-used operations, these application domains can be practically supported across heterogeneous platforms.

1.2 Contributions

In this paper, we make the following contributions:

(1) We introduce a domain specific approach for modelling the run-time characteristics or metrics of heterogeneous computing platforms.

(2) We demonstrate metric modelling in the application domain of computational finance derivatives pricing. Our practical evaluation encompasses a large, diverse workload of 128 computational finance tasks across a heterogeneous computing cluster of 16 CPU, GPU and FPGA platforms across three continents.

(3) We show how the allocation of tasks to platforms can be formulated as a constrained integer programming problem. We demonstrate how the allocation problem can be solved using three distinct approaches: heuristics, machine learning and Mixed Integer Linear Programming (MILP).

(4) We apply the three allocation approaches to both synthetic and real world heterogeneous task and platform data. We show that while heuristics provide acceptable results, machine learning and MILP can provide orders of magnitude more efficient task allocations.

1.3 Proposed Methodology

We demonstrate that domain specific abstractions provide a means for characterising computing platforms in a manner that is *meaningful* in the context of that domain, and hence to the domain programmer.

Furthermore, we show how this domain specific characterisation allows for heterogeneous platforms to be evaluated in a coherent manner, allowing for an efficient allocation of work across these resources.

We seek to provide domain programmers such as Julia with the following programming flow, as illustrated in Figure 1:

(1) She specifies her tasks in a domain specific form.

(2) Her tasks are then characterised using domain metrics with respect to the available platforms.

(3) The optimal task allocations that make up the domain metric trade-off space are found automatically.

(4) Julia then selects the desired trade-off from the metric design space.

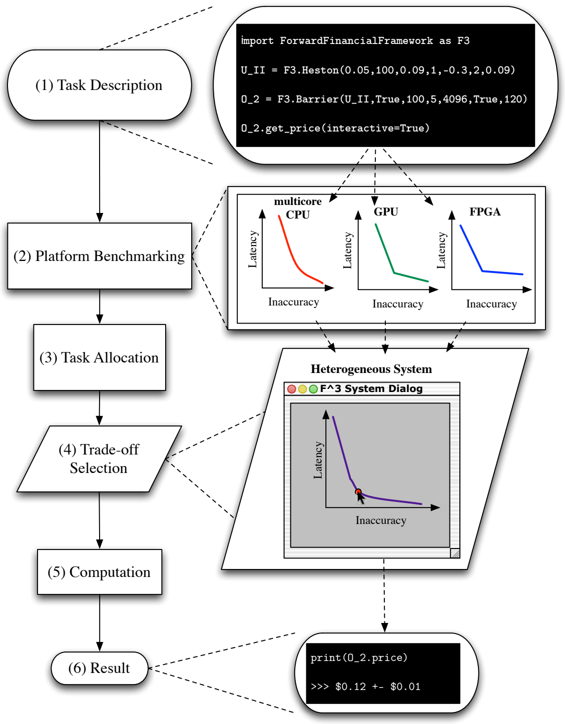
(5) Her workload is then evaluated, using the platforms in accordance with her objectives.

1.4 The Rest of the Paper

In Section 2, we elaborate on the background to the benefits of domain specific abstractions for heterogeneous computing, as well the state-of-the-art with respect to heterogeneous computing characterisation and workload allocation. We then expanded upon our two claims in Section 3: firstly, that domain specific abstractions enable the useful characterisation of heterogeneous platforms, and secondly, that these domain specific metric models can be used in partitioning work across heterogeneous platforms.

Then, in Section 4 we demonstrate the domain specific methodology in practice by applying it to Julia's domain, financial derivatives pricing. We provide a brief overview of the domain and its heterogeneous implementation, after which we describe the latency and accuracy metric models, as well as heuristic, ML and MILP allocation approaches applied. In Sections 5 and 6 we then evaluate our claims in the context of this case study.

Finally we conclude the paper, summarising our major conclusions and lay out suggestions for further work.



```
import ForwardFinancialFramework as F3

U_II = F3.Heston(0.05,100,0.09,1,-0.3,2,0.09)

O_2 = F3.Barrier(U_II,True,100,5,4096,True,120)

O_2.get_price(interactive=True)
```

```
print(O_2.price)

>>> $0.12 +- $0.01
```

Fig. 1: Our proposed domain specific, high-level programming flow for high performance heterogeneous computing.

2 Background

2.1 Domain Specific Heterogeneous Computing

An important finding in recent years is that domain specific abstractions can enable improved performance in a heterogeneous computing context [4], [5], [6]. As alluded to in the introduction, empirical studies of software engineering [1] have found that a small set of algorithmic operations or design patterns within an application domain are executed disproportionately more frequently than others, often following a power law distribution. Indeed, application domains are often identified by grouping these operations together [2]. By supporting the efficient, heterogeneous acceleration of these disproportionately influentially operations, significant gains can be realised automatically for programs restricted to a particular domain.

Previous works have shown domain specific-enabled heterogeneous performance in practice, such as our own use of software application frameworks [6], or domain specific languages, as shown by Chafi et al [4] and Thomas and Luk [5]. The key information yielded by the domain specific abstractions is the implicit dependency relationships between computations, allowing for heterogeneous parallelism to be exploited without programmer intervention.

However putting this approach into practice remains a challenge, requiring system developers with domain expertise to create domain specific abstractions [5], [6] that support heterogeneous execution. Chafi et al's [4] approach advocates the use of language virtualisation, providing both a framework for creating implicitly parallel domain specific languages as well as a dynamic run-time for running applications created using such languages.

2.2 Task Characterisation and Allocation on Heterogeneous Platforms

The problem of characterising and allocating computational tasks to heterogeneous computing platforms has been widely studied for almost 40 years [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18].

2.2.1 Task Characterisation

As identified by Braun et al [7], characterising the execution of tasks upon heterogeneous computing platform is comprised of three interrelated activities:

Task Profiling: identifies the atomic (i.e. indivisible) tasks that comprises the current application. These tasks can then be further qualified by performing analysis or profiling of the task code. A key insight from Khokhar et al [8] is that profiling should determine the parallel execution modes possible for the given task. An increasingly popular approach is to require the programmer to identify the parallel execution modes, either through a specially designed API [19] or by embedding this within the language itself [4].

Analytic Platform Benchmarking: identifies the capabilities of the heterogeneous computational platforms available. Another insight from Khockar et al [8] is that this process details how well the platform supports different parallel execution modes. A heterogeneous benchmark such as Rodinia [20] could be used for this purpose, or a representative subset of the current tasks.

Task-Platform Characterisation: synthesises the data from the two previous activities, which results in models of how the specified tasks will execute upon the available resources. Grewe's work [14] illustrates how a sophisticated machine learning-based approach can be used to do so.

As described in the next subsection, the last activity is usually not distinguished from allocating of tasks to platforms [14], [19]. We argue that maintaining this separation is useful, as it allows for the quality of the characterisation activities to be evaluated independently from the allocation approach that is being used.

2.2.2 The Allocation Problem

When considering the allocation of tasks to heterogeneous computing resources, the general scenario considered in the literature, i.e. [9], [10], [11], [14], [15], [16], [17], [18] is a set of independent or atomic tasks being partioned across or allocated to multiple heterogeneous platforms. It is assumed that a task will block a platform for its duration, i.e. occupy the computing resource completely. It is also commonly assumed that the allocation is being performed statically, in advance of the execution of any of the tasks.

In this general problem formulation, the general objective is to minimise the makespan. The makespan is the latency between when the first task is initiated until the last result returned for the task set.

As described above, minimising the makespan with *a priori* knowledge of the execution time of atomic tasks is a well studied problem. As we shall show in Section 3, as others have [17], this problem can be expressed formally as a 0-1 integer linear programming problem which has famously been shown to be NP-complete by Karp [21].

2.2.3 Allocation Approaches

Surveying the literature, there are three categories of suggested approaches to the allocation problem described above:

Heuristic [9], [11], [16]: a simple algorithmic rule is applied to allocate tasks to the available resources. Under specified circumstances such a rule might achieve a provably optimal allocation of tasks, and there is usually a lower bound on the quality of the solution relative to the optimal solution.

Machine Learning [14], [15]: a feasible task-platform allocation is improved using machine learning techniques such as Danzig's Simplex algorithm, simulated annealing or genetic algorithms. At worst these techniques can confirm the quality of the starting solution.

Integer Linear Programming [10], [17], [18], [22]: the problem can be formulated as a constrained integer program that can be solved using linear optimisation techniques. In addition to applying standard optimisation heuristics, a dual formulation of the problem can be used to prove the optimality of the solution.

2.2.4 Analysis

Generally heuristic approaches have been the most studied in the context of heterogeneous computing. Braun's comprehensive study [9] found that simpler heuristics achieve better results than more complex ones for the general case. This suggests that the truly optimal approach is case-specific, dependent upon the dynamics between the task and platforms concerned, and so the more complex an allocation approach, the more likely it is to map better to certain configurations than others.

ILP appears to be an understudied approach, usually applied only in environments of pressing resource constraint [10]. This lack of attention is likely due to the NP-hard complexity of mixed integer linear programs and the NP-complete complexity of binary valued programs.

However considerable progress has been made in ILP in the last three decades [23], and hence we believe that this approach is now practical for run-time allocation [18], as do others [17]. A key insight is that an external measurement of solution quality is desirable so that a high quality solution that is not necessarily provably optimal can be identified.

3 Domain Characterisation & Allocation

In this section we elaborate on our claims that a domain specific approach to heterogeneous computing allows for both the useful characterisation of task upon heterogeneous platforms, and in turn, an efficient allocation of those tasks to platforms. To illustrate our explanation, in this section we use examples from the domains of image filtering and linear algebra arithmetic.

In Section 4, we apply our domain specific approach to the financial derivatives pricing domain.

3.1 Characterising Tasks upon Platforms

By useful characterisation, we mean actionable, i.e. the domain specific approach enables predictive modelling of the run-time characteristics of domain tasks upon a wide range of heterogeneous platforms. Characterisation would be useful to domain programmers such as Julia as it allows for the static comparison of different platforms which, as we will show in the next subsection, is critical in the efficient allocation of task workloads.

However, this domain specific characterisation is a contribution in its own right because it relates tasks and platforms using the fundamental concepts of the application domain. By modelling the task-platform relationship using domain metrics, the computational design space is made accessible to anyone working within that domain.

These models allow domain programmer to balance their objectives in terms they understand.

3.1.1 Domain Metrics

To find the computational design space for a task or group of tasks within an application domain, we first need to know what the dimensions of that design space should be, i.e. the quantitative measurements used within the domain. We define these quantitative characteristics of the domain *metrics*.

While the actual metric used will vary from domain to domain, all fall into one of four categories:

- *Latency*: the time between initiation and completion.
- *Throughput*: the rate at which the task is completed.
- *Quality*: the degree to a quantifiable goal is achieved.
- *Resource Use*: the resources used to complete the task.

For example, within the domain of image filtering, latency could be measured in the seconds required to filter an image, while throughput could be the number of images processed per second.

In the linear algebra arithmetic domain, quality might be measured as the unit of least precision in the calculations performed, while the resource use might be expressed using the average monetary cost per matrix arithmetic operation.

3.1.2 Metric Models

To predict metrics, we require models for how the task inputs map to the domain metrics on the target platform. We formalise these models in (1): we seek model functions that map p real-valued inputs to domain functions to m real-valued metric values, i.e. $\vec{F} : \vec{P} \to \vec{M}$, where $\vec{F}$ is the domain function model, $\vec{P}$ the inputs and $\vec{M}$ the metric values.

$$\begin{aligned} \vec{F} = (f_1, f_1, \cdots, f_m) : \vec{P} \to \vec{M} \quad & \vec{P} \in \mathbb{R}^p, \vec{M} \in \mathbb{R}^m, \\ f_k(\vec{P}) = M_k \qquad & k = 1, 2, \ldots, m. \end{aligned} \tag{1}$$

As the application domain identifies in advance those operations which are disproportionately used, a function for mapping $\vec{P}$ to $\vec{M}$ of key domain functions for heterogeneous platforms can be found in advance.

For example, in the domain of image filtering, a candidate function for modelling would be the convolution operation used in all image filtering operations. In linear algebra arithmetic, the arithmetic operations would be modelled.

3.1.3 Domain Variables and Parameters

We refine the metric models further in (2), $\vec{P}$ defines all possible input vectors to the domain specific operation. This space can be divided into two disjoint subsets, valid ($\vec{P_v}$) and invalid ($\vec{P_i}$) inputs. $\vec{P_i}$ are all of the inputs that will return a result that violates the correctness of the function as defined within the domain.

$$\vec{P} = \vec{P_i} \cup \vec{P_v} \quad \vec{P_i} \cap \vec{P_v} = \emptyset. \tag{2}$$

For example, in the image filtering domain, when applying a uniform blur to an image, the set of inputs that define a non-uniformly weighted filter would be within $\vec{P_i}$ for that function. $\vec{P_v}$ is thus all of those inputs which return a valid result, representing the design space for that function.

By supplying the definition of "correctness", the application domain makes explicit what input elements may be varied without affecting the correctness of the result. For example, in the linear algebra arithmetic domain, an input which specifies the maximum number of elements computed in parallel can be varied without affecting the correctness of the result.

We define those input elements which can be varied as *domain variables* and those that cannot as *domain parameters*. In our formalism, the domain definition identifies the subset of $\vec{P}$ upon which membership of $\vec{P_v}$ is defined.

3.1.4 Identifying and Populating Metric Models

The formalism above provides the criteria for potential metric model functions, however for each domain function there are an infinite number of possible metric model functions. When choosing one, we found that the simplest models to be the most broadly applicable.

For example, in the linear algebra arithmetic domain, a hypothetical metric model for latency of matrix-matrix addition operation might be expressed as the product of the size in the two matrices concerned (N) and the time per element-wise operation on that platform (α), i.e.

$$f_L(N) = \alpha(N).$$

Similarly, in image filtering, the cost metric for applying a certain filter might be the cost per second of the platform (β) multiplied by the latency of the image processing ($f_L(S)$), i.e.

$$f_C(S) = \beta f_L(S).$$

As the structure of $\vec{F}$ is deterministic, an online benchmarking approach can be used to find the task and platform-specific metric model coefficients. We suggest a benchmarking procedure to generate a set of domain variable and metric values i.e. $\mathbb{R}^{b \times p}$ and $\mathbb{R}^{b \times m}$, where b is the number of benchmarking iterations. The benchmarking data can then be used to to solve for the metric model function's coefficients.

We found weighted least squares regression to be effective in solving for the metric model coefficients. By using the variable benchmark values as weights, we reduced the impact of "noise" present in metric measurements.

3.2 Allocating Task to Platforms

While the characterisation described in the previous subsection is useful when considering a heterogeneous platform in isolation, it is less helpful when faced with a cluster of heterogeneous platforms that can be used cooperatively. In this subsection we address how multiple computational domain metric model functions can be combined so as to create a unified, efficient design space.

3.2.1 The General Allocation Problem

We begin by expressing the makespan minimisation problem, as described in Section 2.2.2, as a binary valued integer linear program in (3).

Each non-zero element of the binary *allocation* matrix ($\boldsymbol{A}$) represents an allocation of one of the τ tasks in a workload to one of the μ platforms, i.e. if $A_{i,j} = 1$, then task j has been allocated to platform i. The relative latency matrix ($\boldsymbol{L}$) gives the latency of each task upon each platform. Hence, similar to $A_{i,j}$, $L_{i,j}$ is the estimated relative latency of task j upon platform i.

$$\begin{aligned} \underset{\boldsymbol{A}\in\{0,1\}^{\mu\times\tau}}{\text{minimise}} \quad & G(\boldsymbol{A},\boldsymbol{L}) \quad \boldsymbol{L}\in\mathbb{R}_{+}^{\mu\times\tau}, \\ \text{subject to} \quad & \sum_{i=1}^{\mu} A_{i,j} = 1 \quad j=1,2,\ldots,\tau. \end{aligned} \tag{3}$$

where:

$$\begin{aligned} G(\boldsymbol{A},\boldsymbol{L}) &= \max(\vec{H}(\boldsymbol{A},\boldsymbol{L})), \\ \vec{H}(\boldsymbol{A},\boldsymbol{L}) &= (\boldsymbol{A}\circ\boldsymbol{L})\cdot\boldsymbol{1}. \end{aligned}$$

Reflecting the makespan minimisation problem's objective, we seek to minimise $G(\boldsymbol{A},\boldsymbol{L})$ while ensure that each task is completed, hence the constraint that the sum of each task entry, i.e. a column of $\boldsymbol{A}$, is 1.

This representation contains contains two reduction functions: firstly, the *task latency reduction* ($\vec{H}(\boldsymbol{A},\boldsymbol{L})$), that is given by the element-wise multiplication or Hadamard product ($\boldsymbol{A}\circ\boldsymbol{L}$), dot multiplied by a vector of ones ($\boldsymbol{1}$); secondly, the *platform latency reduction* ($G(\boldsymbol{A},\boldsymbol{L})$), that finds the maximum latency amongst the platforms for that allocation.

These reduction functions map the allocation and task latency matrices ($\boldsymbol{A},\boldsymbol{L}$) to a vector of platform latencies, with an entry for each platform, and by which the vector of platform latencies are mapped to a scalar makespan value.

We now generalise this program, making use of the notion of domain metric models given in (1). We assume that the valid variables $\vec{P}_v$ for each of the μ platforms are already known or can be easily approximated for each of the τ tasks. In (4) we seek an allocation ($\boldsymbol{A}$) so that we optimise the metric (M_k) for all tasks, as mapped by the task and platform reduction functions ($\boldsymbol{F}_k(\boldsymbol{A},\boldsymbol{P_v})$, $\vec{H}_k(\boldsymbol{A},\boldsymbol{P_v})$ and $G_k(\boldsymbol{A},\boldsymbol{P_v})$) into a scalar value.

$$\begin{aligned} \underset{\boldsymbol{A}\in\{0,1\}^{\mu\times\tau}}{\text{optimise}} \quad & G_k(\boldsymbol{A},\boldsymbol{P_v}) \quad \boldsymbol{P_v}\in\mathbb{R}^{\mu\times\tau\times p}, \\ \text{subject to} \quad & \sum_{i=1}^{\mu} A_{i,j} = 1 \quad j=1,2,\ldots,\tau. \end{aligned} \tag{4}$$

where:

$$\begin{aligned} & G_k(\boldsymbol{A},\boldsymbol{P_v}) : \vec{M}_k \to M_k \quad \vec{M}_k\in\mathbb{R}^{\mu}, M_k\in\mathbb{R}, \\ & \vec{H}_k(\boldsymbol{A},\boldsymbol{P_v}) : \boldsymbol{M_k} \to \vec{M}_k \quad \boldsymbol{M_k}\in\mathbb{R}^{\mu\times\tau}, \\ & \boldsymbol{F_k}(\boldsymbol{A},\boldsymbol{P_v}) : (\boldsymbol{A},\boldsymbol{P_v}) \to \boldsymbol{M}_k. \end{aligned}$$

3.2.2 Splitting the Atomicity of Tasks

Similar to the problem of heterogeneous characterisation, knowledge from the application domain can help find an efficient solution of the allocation problem. As structure of tasks is known in advance, the degree of parallelism within a task is known. As a result, allocation approaches can incorporate this information so as to allow for a task to be divided into subtasks while still providing a correct result. Making parallelism explicit enables a greater degree of work sharing between distributed computing resources, as discussed in Section 2.

In this formulation, if the degree of parallelism is sufficiently large, this allows the elements of the allocation matrix, $\boldsymbol{A}$, to be "relaxed", i.e. to be real-valued, and hence, the problem becomes linear and more tractable, as expressed in (5).

$$\begin{aligned} \underset{\boldsymbol{A}\in\mathbb{R}_{+}^{\mu\times\tau}}{\text{optimise}} \quad & G_k(\boldsymbol{A},\boldsymbol{P_v}) \quad \boldsymbol{P_v}\in\mathbb{R}^{\mu\times\tau\times p}, \\ \text{subject to} \quad & \sum_{i=1}^{\mu} A_{i,j} = 1 \quad j=1,2,\ldots,\tau. \end{aligned} \tag{5}$$

3.2.3 Multimetric Pareto Surfaces

As the metrics under consideration are also known ahead of execution, additional constraints may be added to the optimisation program for every other metric being considered (M_x), as described in (6). This program requires that the allocation also satisfies all of the metric values specified in addition to optimising M_k.

$$\begin{aligned} \underset{\boldsymbol{A}\in\mathbb{R}_{+}^{\mu\times\tau}}{\text{optimise}} \quad & G_k(\boldsymbol{A},\boldsymbol{P_v}) \quad \boldsymbol{P_v}\in\mathbb{R}^{\mu\times\tau\times v}, \\ \text{subject to} \quad & \sum_{i=1}^{\mu} A_{i,j} = 1 \quad j=1,2,\ldots,\tau, \\ & G_x(\boldsymbol{A},\boldsymbol{P_v}) = G_x \quad x\neq k, x=1,2,\ldots,m. \end{aligned} \tag{6}$$

The multimetric optimisation program can be used to generate a Pareto surface, representing the heterogeneous computing platforms in terms of domain metric trade-offs. These trade-offs are achieved by changing the allocation of tasks to platforms.

For the metric Pareto surface to be populated, a range of values are required for all metrics that satisfy the program. This ranges of metrics can be found using the ϵ-constraint method, as described by Kirlik and Sayın [24].

This multimetric Pareto surface represents the culmination of our application of domain knowledge to heterogeneous computing. Our domain specific approach abstracts the allocation of task to platforms as the balancing of domain metrics. Hence, programmers such as Julia would be seamlessly able to use the capabilities of their heterogeneous platform by merely balancing their objectives.

4 Case Study: Derivatives Pricing

In this case study, we show how the domain specific modelling and allocation approach that we developed in the previous section can be applied to a new domain. We use the derivatives pricing domain, as we have done in other work [18], in the broader area of computational finance, given its importance in global commerce.

We first describe the derivatives pricing domain, we then introduce the metric models of latency and accuracy that we use in our evaluation, and finally how the associated allocation problem can be solved using three different methods.

4.1 Derivative Pricing Application Domain

In this subsection we introduce the computational finance application domain, derivatives pricing, that we use as an example in the explanation of our approach and experiments to justify our claims. First, we describe derivatives pricing in general, and then define it as a computational domain. Finally, we describe an implementation of the domain.

4.1.1 Derivatives Pricing Background

Computational finance is an important activity in modern commerce. The problems in the area are concerned with the quantitative measurement of uncertainty or risk. Derivatives pricing is one of the largest activities in this area, with ≈ \$100 trillion of derivatives products currently active. Derivative pricing is also computationally intensive, and as a result is a major consumer of high performance computing, including multicore CPUs and GPUs.

An example of a derivative is an option contract. An option is contract where a holder pays a premium to the writer in order to obtain rights with regards to an underlying, an asset such as a stock or commodity. This right either allows the holder to buy or sell the underlying at a defined strike price at a defined exercise time. The holder has bought the *right* to exercise the transaction if they so choose, and is in no way obligated to so. In derivatives pricing, the intrinsic value of the option is the payoff, the difference between the strike price and spot price of the underlying at the exercise time, or zero, whichever is higher [25].

The popular Monte Carlo technique for option pricing uses random numbers to create scenarios or simulation paths for the underlying based upon a model of its spot price evolution. The average outcome of these paths is then used to approximate the payoff [25], as illustrated in Figure 2. Although computational expensive, this technique is robust, capable of tolerating underlying models with many more stochastic variables than competing methods [5], [25]. Another advantage is that it is amenable to parallel execution. In fact, Monte Carlo is the canonical "Embarrassingly Parallel" algorithm [26].

4.1.2 Application Domain

We now describe derivative pricing as an application domain in terms of types and functions.

The Underlying and Derivative Types: the data in an option pricing task may be subdivided into two components, the derivative contract which is being priced and the underlying asset from which that derivative derives its value. The underlying encapsulates the probabilistic model, such as the

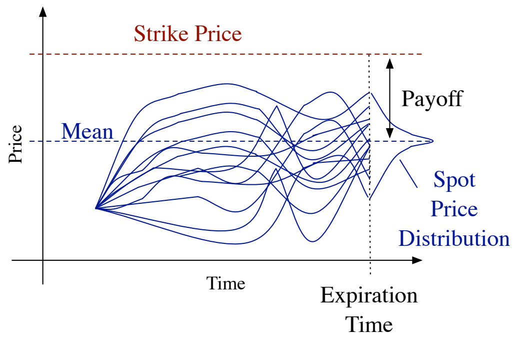


Fig. 2: Overview of Monte Carlo derivatives pricing.

Black-Scholes or Heston, being used to model the behaviour of the asset. The derivative embodies the details of the option contract both during the lifetime of the option as well at its expiration.

The communication within a task can be formulated as a directed, acyclic graph, in which underlyings feed their prices to the derivatives which depend upon them.

Pricing Function: The option pricing domain's sole function is finding the value of a type. Hence, the pricing function is typically only applied to the derivative type, as by definition an underlying type can provide its price at any point in time. Different techniques such as Monte Carlo or Tree-based methods could be used to implement the pricing function, provided the end result is the price of the derivative under consideration.

4.1.3 Domain Implementation

We now describe the Forward Financial Framework[1](F^3), an open source, financial domain application framework that we have developed, that implements the derivatives pricing domain.

Task description: F^3 is implemented at the high level as a Python library [6]. A domain user, a financial engineer such as Julia, may use F^3's classes to describe their derivatives pricing computations. There are three fundamental base classes that mirror the key concepts in the domain: derivatives, underlyings and solvers.

The underlying and derivative objects capture the attributes and behaviours of the underlying and derivative types as described above. The solver class is a collection for the derivatives that the programmer wishes to price as well as the platforms they wish to use.

Heterogeneous Implementations: The solver class supports three behaviours upon heterogeneous platforms: code generation, compilation and execution. F^3 uses a wide array of back-end technologies: multicore CPUs using POSIX C; GPUs, Xeon Phi coprocessors and Altera FPGAs using the OpenCL standard [27]; Maxeler FPGAs using Maxeler's MaxJ.

All of the platform back-ends use a host-accelerator configuration, where a high performance coprocessor or subsystem is managed by a commodity CPU host. Communication between F^3 and platforms use the Secure Shell (SSH) protocol, allowing for tasks to be executed on remote platforms via TCP/IP networks.

1. https://github.com/Gordonei/ForwardFinancialFramework

4.2 Financial Latency and Accuracy Metric Models

As described in the previous subsection, for our example domain, pricing is the only function required. In this subsection, we develop the metric models, as per (1), for the metrics of latency, (7), and price accuracy, (8), for the pricing function as implemented using the Monte Carlo algorithm in F^3. This is in contrast to our other work, where we developed a financial cost metric model [18].

4.2.1 Latency Model

The latency between when a pricing operation is initiated and when it returns a price is fundamentally important within the financial domain [25]. This is because the time at which prices are received affects how traders use those prices. Minimising the latency of the pricing operation is desirable, as this confers first-mover advantage.

We have used a simple, linear latency model in (7), a function of a single domain variable, the number of simulation paths (n), i.e $n \in \mathbb{Z}, n \in \vec{P}_v$. The linear nature of the model reflects the $O(N)$ complexity of the Monte Carlo Algorithm. The model's coefficient (β) translates to the time spent per Monte Carlo path. Similarly, γ, the constant component of the latency metric model, captures the fixed time spent initialising the computation, as well as any time spent communicating the task to, and returning the result from, the target platform.

$$f_L(n) = \beta n + \gamma. \tag{7}$$

4.2.2 Accuracy Model

In the financial domain, the accuracy of a computed price is expressed in probabilistic terms. When using the Monte Carlo technique, the size of 95% confidence interval, as measured in currency of pricing (i.e. \$) is used. The accuracy measure is the size of the finite interval around the computed price for which there is a 95% confidence that the true value lies within that interval. As small a confidence interval as possible is desired, as this means less variance in the price has to be accounted for.

The accuracy model that we used is based upon the convergence of the Monte Carlo algorithm, which is given by the inverse square root of the paths, scaled by a coefficient (α). The model is given in (8).

$$f_C(n) = \alpha n^{-\frac{1}{2}}. \tag{8}$$

4.2.3 Combined Model

To relate the two domain metrics of latency and accuracy, we can solve for n and use it to relate (7) and (8) into a trade-off between the latency and accuracy (c), as given in (9).

$$f_L(c) = \delta c^{-2} + \gamma. \tag{9}$$

where:

$$\delta = \beta\alpha^2.$$

4.3 Derivative Pricing Task Allocation

We can now formulate the allocation problem using the derivative pricing metric models from the previous subsection, as well as outline three approaches for solving the problem.

4.3.1 Reformulating the Allocation Problem

In (10) the unified domain metric model described in (9) has been applied to the general, constrained allocation problem formulated in (6). The vector $\vec{c}$ gives the required accuracies for the tasks, while $\boldsymbol{\gamma}$ is the task-platform constant matrix. Similarly, $\boldsymbol{\delta} : \vec{c}^2$ is the element-wise division of the delta coefficients by the required accuracies of the tasks. In this case, we have not to had to add an additional accuracy constraint, as the unified metric model has already captured this constraint.

$$\begin{aligned} \underset{\boldsymbol{A}\in\mathbb{R}_+^{\mu\times\tau}}{\text{minimise}} \quad & G_L(\boldsymbol{A}, \vec{c}) \quad \vec{c} \in \mathbb{R}_+^{\tau} \\ \text{subject to} \quad & \sum_{i=1}^{\mu} A_{i,j} = 1 \quad j = 1, 2, \ldots, \tau. \end{aligned} \tag{10}$$

where:

$$\begin{aligned} G_L(\boldsymbol{A}, \vec{c}) &= \max(\vec{H}_L(\boldsymbol{A}, \vec{c})), \\ \vec{H}_L(\boldsymbol{A}, \vec{c}) &= (\boldsymbol{\delta} : \vec{c}^2 \circ \boldsymbol{A} + \boldsymbol{\gamma} \circ \lceil \boldsymbol{A} \rceil) \cdot \mathbf{1}, \\ & \boldsymbol{\delta} \in \mathbb{R}_+^{\mu\times\tau}, \boldsymbol{\gamma} \in \mathbb{R}_+^{\mu\times\tau} \end{aligned}$$

An important feature of the formulation given in (10) is its non-linearity as a result of the ceiling function in $\vec{H}_L(\boldsymbol{A}, \vec{c})$. This reflects (7) and (9), as there is a constant value for each platform-task entry, regardless of the scale of the allocation.

4.3.2 Proportional Allocation Heuristic

The first allocation approach, the proportional allocation heuristic, is given in (11). The heuristic allocates tasks inversely proportionally to the individual makespans of all of the platforms, attempting to balance tasks according to the relative capabilities of the different platforms.

$$\vec{A}_{i,j} = \left(\vec{L}_i \sum_{o=1}^{\mu} \frac{1}{\vec{L}_o} \right)^{-1} \quad i = 1, 2, \ldots, \mu, j = 1, 2, \ldots, \tau. \tag{11}$$

where:

$$\vec{L} = \vec{H}_L(\mathbf{1}, \vec{c}).$$

The heuristic only require an estimate of the relative latency of all tasks upon each platform. The proportional allocation heuristic works well provided the elements of $\boldsymbol{\gamma}$ are significantly smaller than the elements of $\boldsymbol{\delta} : \vec{c}^2$ for all platforms. If not, the tasks' cumulative constants dominate each platform's makespan, regardless of allocation. Theoretically, if there were no constant components, i.e. no setup time, then this heuristic would return the optimal allocation.

4.3.3 Machine Learning Allocation

The second approach uses the heuristic as a starting allocation of tasks. The platform reduction function $G_L(\boldsymbol{A}, \vec{c})$ is then specified as the objective function for a time-constrained, global optimisation algorithm, the simulated annealing algorithm provided in SciPy [28], combined with a "polishing", convex optmisation algorithm, Danzig's Simplex algorithm, also available in SciPy.

As this approach incorporates domain specific platform and task information as well as the heuristic, it should at worst confirm the heuristic, and at best find the most optimal allocation. As we will show in Section 6, the objective function's linearity is a key determinate of the allocation optimality. Furthermore, another significant factor is problem size, as this problem suffers from the curse of dimensionality with respect to both μ and τ.

4.3.4 Mixed Integer Linear Programming Allocation

The MILP approach uses the formulation of the domain allocation problem as the input to an open source, constrained integer programming framework, SCIP [29]. SCIP applies global optimisation techniques as well as a variety of mathematical transformations and heuristics to solve the constrained problem.

SCIP accepts problems in a form very similar to (10), expressed in Zuse Institut Mathematical Programming Language (ZIMPL) [30], which F^3 is capable of generating. However ZIMPL/SCIP does not allow for non-linear objective and constraint functions. This requires the problem to be reformulated as given in (12), adding additional variables (G_L and $\boldsymbol{B}$) and constraints to capture the non-linearities in the problem.

$$\begin{aligned}
\underset{G_L,\boldsymbol{A},\boldsymbol{B}}{\text{minimise}} \quad & G_L \quad G_L \in \mathbb{R}_+, \boldsymbol{A} \in \mathbb{R}_+^{\mu\times\tau}, \boldsymbol{B} \in \{0,1\}^{\mu\times\tau}, \\
\text{subject to} \quad & \sum_{i=1}^{\mu} A_{i,j} = 1 \quad j = 1,2,\dots,\tau, \\
& H_{L,i}(\boldsymbol{A},\vec{c}) \leq G_L \quad \vec{c} \in \mathbb{R}_+^{\tau}, i = 1,2,\dots,\mu, \\
& A_{i,j} \leq B_{i,j} \quad i = 1,2,\dots,\mu, j = 1,2,\dots,\tau.
\end{aligned} \tag{12}$$

Where

$$\vec{H_L}(\boldsymbol{A},\vec{c}) = (\boldsymbol{\delta} : \vec{c}^2 \circ \boldsymbol{A} + \boldsymbol{\gamma} \circ \boldsymbol{B}) \cdot \mathbf{1}.$$

Although binary integer linear programs are known to be NP-complete [21], there has been progress in solving these problems efficiently [23].

5 Evaluating Derivative Pricing Metrics

In this section, we evaluate our claim that the derivatives pricing metric models are able to characterise tasks on platform.

To do so we need to evaluate the following two properties for the latency and accuracy models using a large, diverse set of platforms and tasks: *Incorporation*: When provided with additional information, the domain metric model predict the run-time value of that domain metric more accurately. *Extrapolation*: For a given amount of benchmarking, the domain metric values predicted by the models remains reasonably close to those seen at run-time for an increasing problem size.

To assess the degree to the properties were achieved, we measured the relative error (E_k) as given in (13), where the absolute difference between the predicted metric value ($f_k(n)$) and the run-time value ($\hat{f}_{k,n}$) is divided by the run-time value. The run-time metric value is measured when the task is run with the specified number of paths (n).

$$E_k = \frac{\left| f_k(n) - \hat{f}_{k,n} \right|}{\tilde{f}_{k,n}} \tag{13}$$

TABLE 1: Evaluation workload of 128 derivative pricing tasks. Underlying types are Black-Scholes (*BS*) and Heston (*H*) model-based. Derivative types are Asian (*A*), Barrier (*B*), Double Barrier (*DB*), Digital Double Barrier (*DBB*) and European Options (*E*).

Task Designation	Number	Underlying	Option	Computational Operations (kFLOP / path)
BS-A	10	BS	A	139.267
BS-B	10	BS	B	139.266
BS-DB	10	BS	DB	143.360
BS-DDB	5	BS	DDB	143.361
H-A	25	H	A	319.492
H-B	29	H	B	319.491
H-DB	29	H	DB	323.585
H-DDB	5	H	DDB	323.586
H-E	5	H	E	315.395

5.1 Experimental Setup

5.1.1 Derivatives Pricing Tasks

Table 1 provides an overview of the 128 option pricing tasks that were used to evaluate the financial domain metric models. In addition to the types of underlying and derivatives used, the total amount of computational work for each task is specified.

The domain parameters for the pricing task operations, such as the proprieties of underlying model, were generated using uniform random numbers within the values of the Kaiserslautern option pricing benchmark [31]. We used a rejection procedure to keep the relative complexity of the pricing tasks within an order of magnitude.

5.1.2 Heterogeneous Platforms

An overview of the heterogeneous platforms that we used are described in Table 2. The first class of platform heterogeneity is device type and manufacturer - we used a wide array of multicore CPUs, GPU and FPGA-based computational platforms from a variety of vendors. The other is the diversity of interconnections used between the computational platforms, achieved with varied geographic locations.

The computational characteristics of the platforms are also described in Table 2. We describe the compute capabilities of the experimental platforms using the Kaiserslautern option pricing benchmark [31] and the Network Round-trip Time (RTT) as measured by the *ping* network utility.

As the Monte Carlo algorithm being used is amenable to parallel execution, it is unsurprising that GPUs provide the best application performance, although an important caveat is that these performance figures are of implementations produced by F^3. A prominent data-point in terms of network latency is the Remote Server and Phi, which have orders of magnitude longer communication times than the other platforms due to being located in Cape Town, South Africa.

We expect the compute capabilities of platforms to determine the coefficient of the latency models (β) while the network latency will determine the constant coefficient, (γ).

TABLE 2: Overview of Experimental Heterogeneous Computing Platforms

Device Category	Device Designation	Device Vendor	Device Name	Network Location	Geographic Location	Application Performance (GFLOPS)	Network Round-trip Time (ms)
CPUs	Desktop	Intel ®	Core ® i7-2600	Localhost	ICL, London, UK	5.916	0.024
	Local Server	AMD ®	Opteron ® 6272	LAN	ICL, London, UK	27.002	0.380
	Local Pi	ARM ®	11 76JZF-S	LAN	ICL, London, UK	0.049	2.463
	Remote Server	Intel ®	Xeon ® E5-2680	WAN	UCT, Cape Town, ZA	11.523	3300.000
	AWS Server EC1	Intel ®	Xeon ® E5-2680	WAN	AWS, USA East Region	12.269	88.859
	AWS Server EC2	Intel ®	Xeon ® E5-2670	WAN	AWS, USA East Region	4.913	88.216
	AWS Server WC1	Intel ®	Xeon ® E5-2680	WAN	AWS, USA West Region	12.200	157.100
	AWS Server WC2	Intel ®	Xeon ® E5-2670	WAN	AWS, USA West Region	4.926	159.578
	GCE Server	Intel ®	Xeon ®	WAN	GCE, USA Central Region	6.022	111.232
GPUs	Local GPU 1	AMD®	FirePro ® W5000	LAN	ICL, London, UK	212.798	0.269
	Local GPU 2	Nvidia®	Quardo ® K4000	LAN	ICL, London, UK	250.027	0.278
	Remote Phi	Intel®	Xeon Phi ® 3120P	WAN	UCT, Cape Town, ZA	70.850	3300.000
	AWS GPU EC	Nvidia®	Grid ® GK104	WAN	AWS, USA East Region	441.274	88.216
	AWS GPU WC	Nvidia®	Grid ® GK104	WAN	AWS, USA West Region	406.230	159.578
FPGAs	Local FPGA 1	Xilinx®	Virtex ® 6 475T	LAN	ICL, London, UK	114.590	0.217
	Local FPGA 2	Altera®	Stratix ® V D5	LAN	ICL, London, UK	161.074	0.299

5.2 Model Error Results

5.2.1 Latency Model

The latency model results are given in Figures 3 and 4. The latency models were evaluated on a per platform basis, as well as the geometric mean of the three platform categories.

The mean error figures for all of the tasks upon the platform reported, with the error bars being too small to plot. The independent variable is the ratio of the Monte Carlo benchmark vs run-time paths, so as to report on all tasks across each platform.

Figure 3 illustrates that as a longer benchmarking procedure is performed relative to the total fixed run-time of 10 minutes (or 4.69 seconds per task) being predicted by the model, the models became more accurate. Figure 4 shows how the models scale as the run-time prediction target is increased for a fixed benchmarking time of 4.69 seconds per task or 10 minutes in total, and an increasing run-time target. The remote Phi and server models' poor performance are notable data points.

5.2.2 Accuracy Model

The accuracy model results are given in Figures 5 and 6. The accuracy model results are presented as minimum, geometric mean and maximum of the model results within the pricing task categories. Similar to the latency model, the ratio of benchmark to run-time paths is the independent variable.

Figure 5 shows how the accuracy models become increasingly predictive with additional benchmarking. For all of the task categories, the minimum, mean and maximum errors all decrease as more benchmarking is performed. Figure 6 shows how the models remain stable as the run-time target is increased. Hence, similar to the latency model results, the models scale well for more than an order of magnitude.

5.3 Discussion

The incorporation property of the models is demonstrated in Figures 3 and 5. This means that the predictive capability of the models improves as additional information is provided to the models.

As Figures 4 and 6 illustrate, the models also have the extrapolation property. There is a relatively minor increase in latency and accuracy error for run-times considerably longer than the benchmarking time.

The relatively poor latency model performance of the Remote Phi and Server platforms is explained by the benchmarking time being too short to accurately solve for the true coefficient and constant values. This is due to long network round-trip time that both platforms experienced, where the almost all of the benchmarking time is spent on communication.

The tasks with Heston underlyings present a relatively high maximum accuracy error, between 10% and 100%. However, as can be seen by the task category geometric mean these error average out to a considerably lower figure, allowing for these models to still be useful for modelling groups of tasks.

6 Domain Allocation Approach Evaluation

In this section we describe our evaluation of the allocation approaches that make use of domain knowledge provided through the metric models, machine learning and MILP. We first characterise the performance of the domain allocation approaches with respect to problem size and problem non-linearity using synthetic data. We then verify this characterisation by applying the different allocation approaches to the tasks and platforms described in Tables 1 and 2.

We report on the time required by the domain allocation approach algorithms as well as the quality of the solution returned with respect to the proportional allocation heuristic. For an allocation approach to be acceptable, it needs to cope with a wide variety of allocation problems in we what we heuristically define as a reasonable amount of time, 10 minutes, while providing a significant improvement over the allocation returned by the heuristic approach.

6.1 Experimental Setup

6.1.1 Synthetic Data Generation Procedure

Drawing upon Braun et al's work [9], we have used the following procedure $(s(\tau, \mu, \theta_\tau, \theta_\mu, \omega_\tau, \omega_\mu, \psi))$ to generate

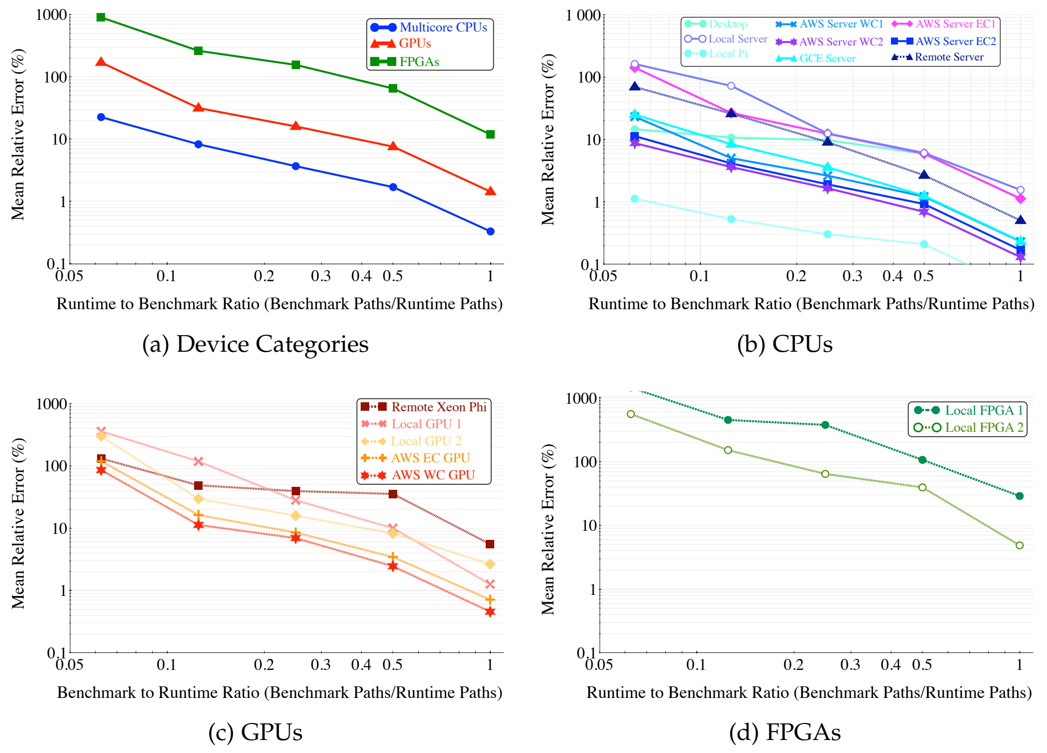


Fig. 3: Error of latency models for a total run-time target of 10 minutes ($\frac{4.69s}{\text{task}}$) and varying benchmark time, evaluating model incorporation.

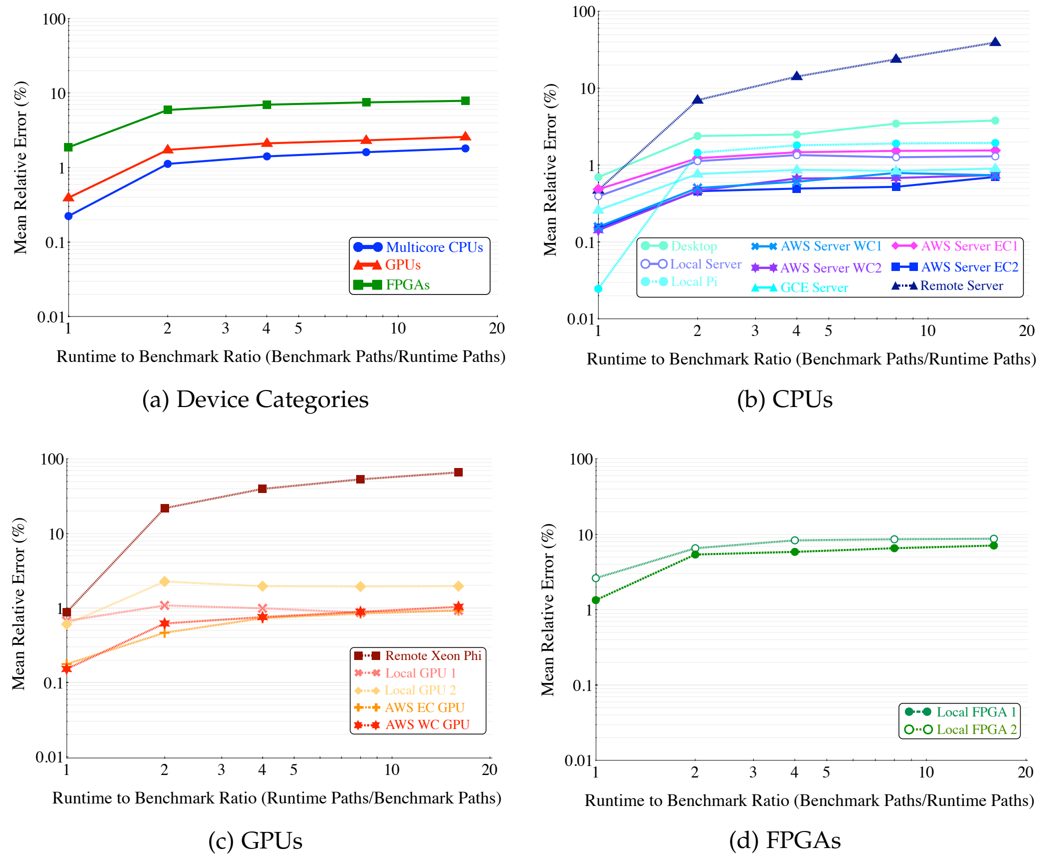
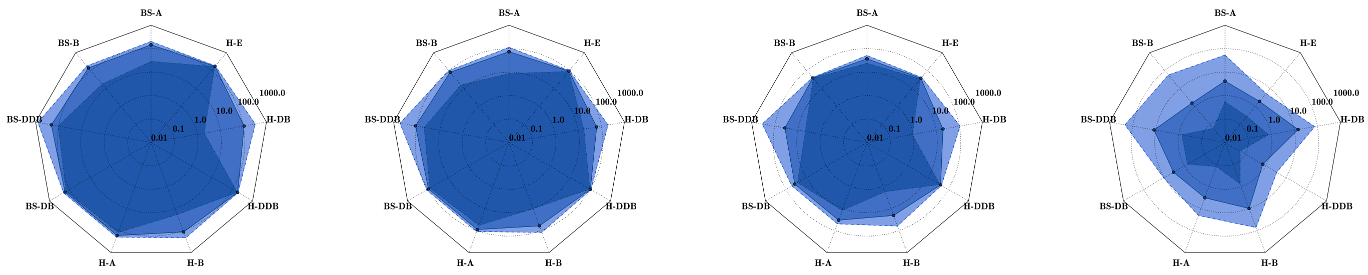
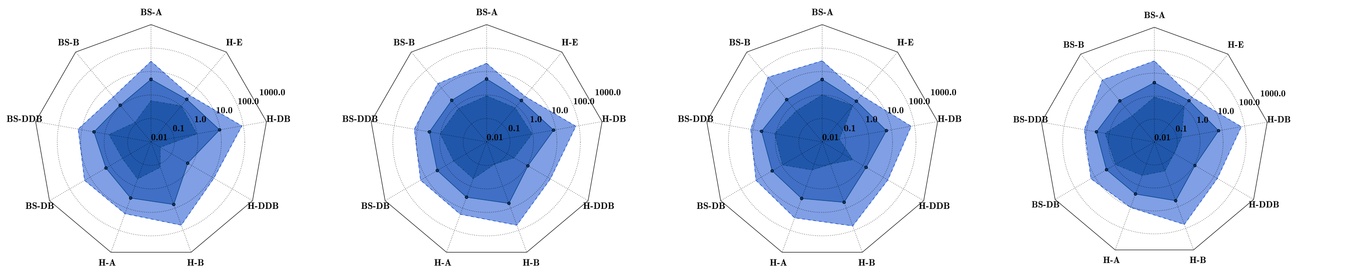


Fig. 4: Error of latency models for a fixed benchmark total time of 10 minutes ($\frac{4.69s}{\text{task}}$) and varying run-time targets, evaluating model extrapolation.



(a) 1:8 (b) 1:4 (c) 1:2 (d) 1:1

Fig. 5: Percent error of accuracy models for a fixed run-time target and varying benchmark time, evaluating model incorporation. Ratio is expressed as *Benchmark Paths* : *Run-time Paths*. The innermost region represents the minimum error, the middle region the geometric mean relative error and the outermost the maximum.



(a) 1:2 (b) 1:4 (c) 1:8 (d) 1:16

Fig. 6: Percent error of accuracy models for a fixed benchmark time and varying run-time, evaluating model extrapolation. Ratios and the ordering of the regions are the same as in Figure 5.

the synthetic co-efficient ($\boldsymbol{\delta}$) and constant ($\boldsymbol{\gamma}$) matrices so to evaluate the different approaches to allocation:

(1) Construct the baseline vector ($\vec{x}$) and initial matrix ($\boldsymbol{Y}$). $\vec{x}$ is τ uniformly distributed integer elements, bounded by the task heterogeneity factor (θ_τ). $\boldsymbol{Y}$, is $\mu \times \tau$ uniformly distributed integer elements, bounded by the platform heterogeneity factor (θ_μ):

$$x_j \in [1, \theta_\tau] \quad j = \{1, 2, \ldots, \tau\},$$

$$Y_{i,j} \in [1, \theta_\mu] \quad i = \{1, 2, \ldots, \mu\}, j = \{1, 2, \ldots, \tau\}.$$

(2) Construct the $\boldsymbol{\delta}$ matrix by multiplying the elements of each row of $\boldsymbol{Y}$ and of $\vec{x}$. i.e.

$$\delta_{i,j} = x_j Y_{i,j} \quad i = \{1, 2, \ldots, \mu\}, j = \{1, 2, \ldots, \tau\}.$$

(3) Sort the first $\tau\omega_\tau$ columns of the $\boldsymbol{\delta}$ matrix, and the first $\mu\omega_\mu$ rows, where ω_τ and ω_μ are the degree of task and platform consistency.

(4) Construct the $\boldsymbol{\gamma}$ matrix by repeating steps 1-3, however then multiply the resulting matrix by the task constant to coefficient ratio (ψ), i.e. the γ to β ratio in the latency metric model.

6.1.2 Procedure Parameter Values

The parameters varied in our experiments, in conjunction with the procedure above are provided in Table 3. The four cases consider a range of different scenarios, from completely homogeneous, consistent to extremely heterogeneous and inconsistent platforms and tasks, using the values from Braun et al's study [9].

TABLE 3: Synthetic task-platform generation parameters. Columns are platform and task heterogeneity (θ_μ, θ_τ), and platform and task consistency (ω_μ, ω_τ).

Case Designation	θ_μ	ω_μ	θ_τ	ω_τ
Hom-Con	10	1.0	100	1.0
Het-Con	100	1.0	3000	1.0
Het-Mix	100	0.5	3000	0.5
Het-Inc	100	0.0	3000	0.0

6.2 Allocation Characterisation Results

6.2.1 Synthetic Data Characterisation

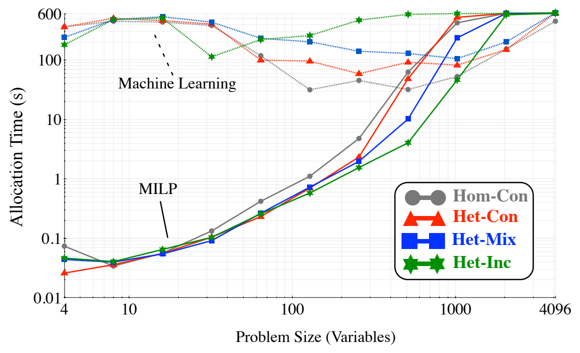
The results of the allocation characterisation can be seen in Figure 7. For the allocation times (Figures 7a and 7b), a timeout of 600 seconds (or 10 minutes) was set, the same time given to the benchmarking described in the previous section.

Similarly for the quality of the solution relative to the proportional heuristic (Figures 7c and 7d), we found that both the MILP and machine learning task allocations' improvements over the heuristic are a function of problem variables and constant to coefficient ratio.

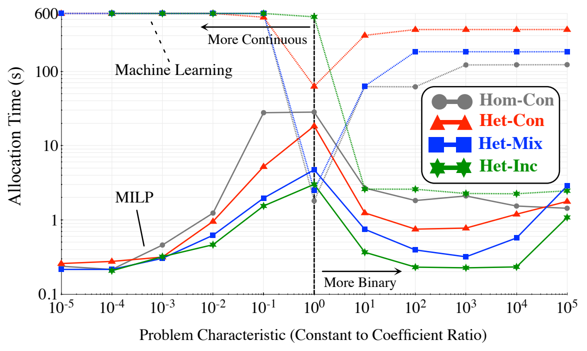
6.2.2 Practical Verification

While the characterisation of the allocation approaches using synthetic data provides insight, we have verified these these results with real platform and task data in Figure 8. We

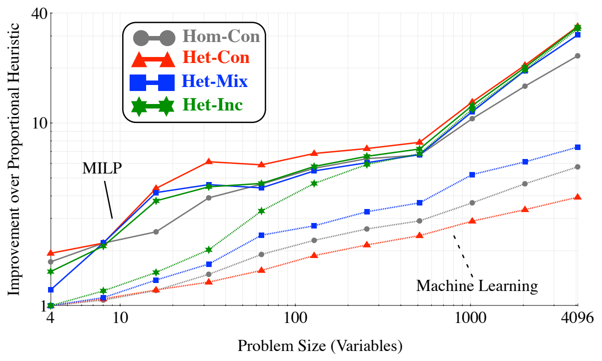
Fig. 7: Characterisation of allocation approaches using synthetic data.



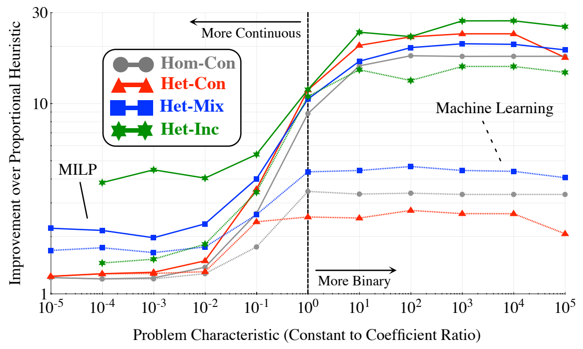
(a) Impact of allocation problem size on allocation time. The constant to coefficient ratio is 1



(b) Impact of problem characteristic on allocation time. The number of variables is 1024

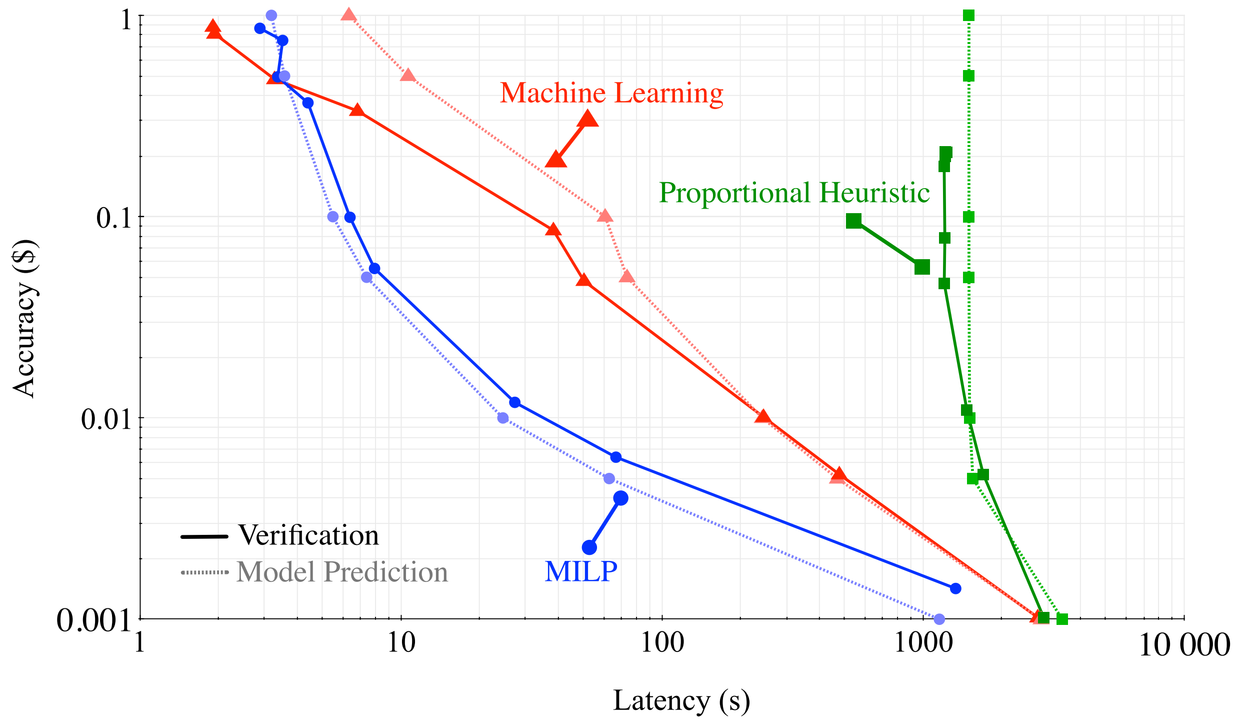


(c) Improvement for varied problem sizes. The constant to coefficient ratio is 1.



(d) Improvement for varied problem characteristics. The number of variables is 1024

Fig. 8: Allocation approaches using heterogeneous platforms from Table 2 and derivatives pricing tasks from Table 1. Smaller latency and accuracy values are better.



put the portfolio of pricing tasks in Table 1 through the allocation approaches for the platforms in Table 2 over a range of accuracies. We then ran the generated task allocations, and measured the domain metrics of latency and accuracy.

6.3 Discussion

Broadly, the machine learning-based task allocation approach was soon limited by the timeout, as evidenced by Figure 7a, while the MILP task allocation's time grows exponentially as a function of the number of variables. As the ratio between the coefficient (β) and constant (γ) component is varied in Figure 7b, there is a peak latency centred around 1, reflecting the considerable linear and non-linear allocation problems that both have to be solved and balanced. The machine learning approaches perform well at this inflection point, while the MILP approach is at its relative worst.

In terms of improvement over the heuristic allocation, Figures 7c and 7d, the trends can be explained by the potential for improvement. The linear trend with respect to problem size is due to the increased potential for improvement that larger problems allow. Similarly, for the constant to coefficient ratio, as the constant becomes more dominant, there is increased scope for improvement as the heuristic is further from its optimal condition.

Figure 8 illustrate that the allocation approaches using the metric models are close to what is measured in reality. The differences between the predictions and runtime measurements are well within the error of the metric models. Furthermore, both the domain knowledge-based machine learning and MILP-based allocation approaches are orders of magnitude more efficient than that suggested by the proportional heuristic for problems with strong non-linear characteristics, i.e. derivatives pricing tasks with 95% confidence intervals greater than $0.005.

7 Conclusion

In this paper we have described and demonstrated in practice that a domain specific approach to heterogeneous computing offers two features beyond portable execution.

Firstly, using image filtering, linear algebra arithmetic and derivatives pricing as example domains, we described how domain metric models derived from the application domain provide a natural means to characterise a task on a heterogeneous platform.

We evaluated the metric models of latency and accuracy for the derivative pricing domain practically. We found that when using an online benchmarking procedure, these domain metric models incorporate additional information to improve predictions, and extrapolate well as tasks increase in scale.

These metric models are an accessible way to visualise the design space of the available heterogeneous platforms for domain programmers, such as Julia, as illustrated in Figure 9. As is to be expected, when the accuracy requirement is low, constant communication time dominate and the platform makespans are geographically ordered, but when high accuracy is required, the compute dominates and the makespans order according to the measured computational capability of the platforms.

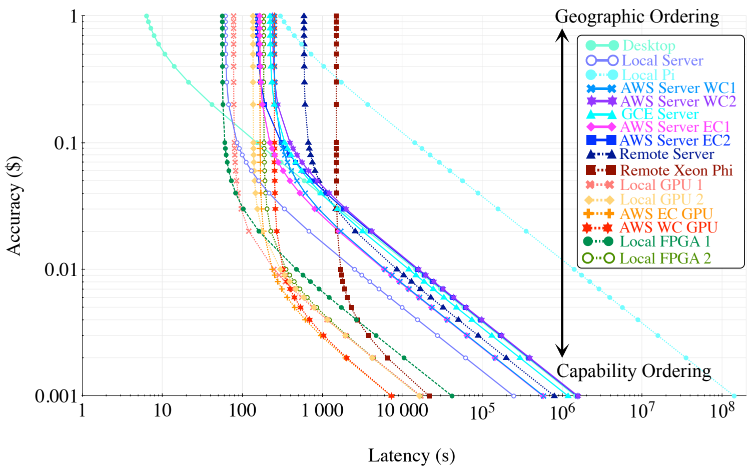


Fig. 9: Heterogeneous task-platform metric curves. Smaller accuracy and latency values are better.

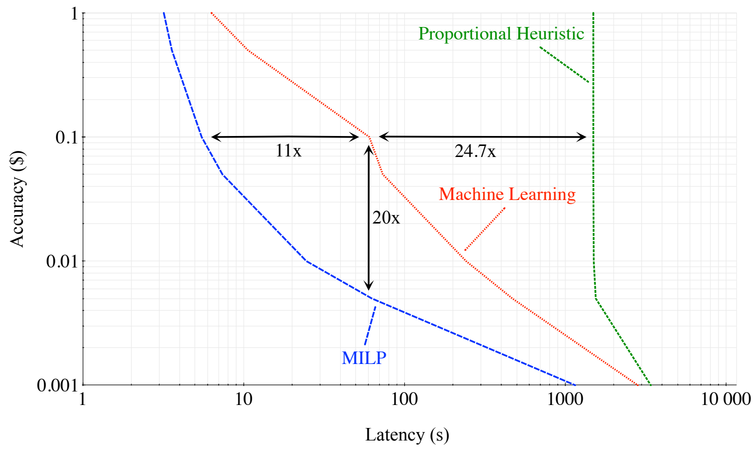


Fig. 10: Differing domain specific task allocation approaches. Smaller accuracy and latency values are better.

Secondly, we described how the metric models for multiple platforms can be combined into a constrained optimisation program. We also showed domain specific knowledge can allow this allocation to be relaxed, transforming the binary problem to a more tractable, mixed integer form.

We described and evaluated three approaches for solving this allocation problem, heuristic, machine learning and MILP. Our evaluation, making use of both synthetic data as well as the derivatives pricing examples, demonstrated that both MILP and machine learning can produce viable task allocation in a practical amount of time whilst outperforming the heuristic approach by up to two orders of magnitude, as illustrated in Figure 10.

Beyond the practical benefits, our domain specific methodology makes distributed, heterogeneous computing platforms *accessible* to domain users, such as Julia in the Introduction. Our approach shows how to abstract away details of implementation into choices about the nature of computational results. We only require that Julia balances her objectives to use heterogeneous computing effectively.

Future Work

Future directions for this work include increasing both the allocation problem sizes as well as the number of platforms utilised. A further direction is in increasing the degree of heterogeneity, both in terms of the problems considered as well as more varied computing resources.

Another area of ongoing work is optimisation of the MILP software used. Improvements being considered are seeding the optimiser with the proportional heuristics proposed in this paper, as well as ordering the heuristics applied within the optimiser more carefully.

ACKNOWLEDGMENTS

Conversations with Prof George Constantinides, Dr Victor Magron, Dr Eric Kerrigan, Dr Andrea Suardi, Mr Shane Fleming and Mr Andrea Picciau have been invaluable.

We are grateful for the funding support from the South African National Research Foundation and Oppenheimer Memorial Trust. We have also benefited from the support of the Nallatech, Altera, Xilinx, Intel and Maxeler university programs.

REFERENCES

[1] B. A. Nardi, *A small matter of programming: perspectives on end user computing*. MIT press, 1993.
[2] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages," *ACM SIGPLAN Notes*, vol. 35, no. 6, pp. 26–36, Jun. 2000.
[3] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.
[4] H. Chafi, A. K. Sujeeth, K. J. Brown, H. Lee, A. R. Atreya, and K. Olukotun, "A domain-specific approach to heterogeneous parallelism," in *Proceedings of ACM SIGPLAN Symposium on Principles and Practises of Parallel Programming*, 2011, pp. 35–46.
[5] D. B. Thomas and W. Luk, "A Domain Specific Language for Reconfigurable Path-based Monte Carlo Simulations," in *Proceedings of International Conference on Field-Programmable Technology*, 2007, pp. 97–104.
[6] G. Inggs, D. Thomas, and W. Luk, "A heterogeneous computing framework for computational finance," in *Proceeding of the International Conference on Parallel Processing*, 2013, pp. 688–697.
[7] T. Braun, H. Siegel, and A. Maciejewski, "Heterogeneous Computing: Goals, Methods, and Open Problems," in *High Performance Computing*, ser. Lecture Notes in Computer Science, B. Monien, V. Prasanna, and S. Vajapeyam, Eds. Springer Berlin/Heidelberg, 2001, vol. 2228, pp. 307–318.
[8] A. Khokhar, V. Prasanna, M. Shaaban, and C. Wang, "Heterogeneous computing: challenges and opportunities," *Computer*, vol. 26, no. 6, pp. 18–27, Jun. 1993.
[9] T. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Muthucumaru, A. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, Jun. 2001.
[10] N. Fisher, J. Anderson, and S. Baruah, "Task Partitioning upon Memory-Constrained Multiprocessors." IEEE, 2005, pp. 416–421.
[11] W. W. Chu, L. J. Holloway, M.-t. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," *Computer*, 1980.
[12] F. Berman and R. Wolski, "Application-level scheduling on distributed heterogeneous networks," in *Proc. 1996 ACM/IEEE Conference on Supercomputing*. IEEE, 1996.
[13] R. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: a scheduling framework for heterogeneous computing," *Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 514–521.
[14] D. Grewe and M. F. O'Boyle, "A static task partitioning approach for heterogeneous systems using OpenCL," in *Lecture Notes on Computer Science*, vol. 6601, 2011, pp. 286–305.
[15] Q. Kang, H. He, and H. Song, "Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm," *Journal of System Software*, vol. 84, no. 6, pp. 985–992, Jun. 2011.
[16] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.
[17] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Scalable linear programming based resource allocation for makespan minimization in heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 84, pp. 76–86, Oct. 2015.
[18] G. Inggs, D. B. Thomas, G. Constantinides, and W. Luk, "Seeing Shapes in Clouds: On the Performance-Cost trade-off for Heterogeneous Infrastructure-as-a-Service," in *International Workshop on FPGAs for Software Programmers*, jun 2015.
[19] C.-K. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.
[20] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterisation*. IEEE, Oct. 2009, pp. 44–54.
[21] R. M. Karp, *Reducibility among combinatorial problems*, 2010.
[22] Shiann-Rong Kuang, Chin-Yang Chen, and Ren-Zheng Liao, "Partitioning and Pipelined Scheduling of Embedded System Using Integer Linear Programming," in *International Conference on Parallel and Distributed Systems*, vol. 2. IEEE, 2005, pp. 37–41.
[23] R. Bixby and E. Rothberg, "Progress in computational mixed integer programmingA look back from the other side of the tipping point," *Anneals of Operations Research*, vol. 149, no. 1, pp. 37–41, Jan. 2007.
[24] G. Kirlik and S. Sayın, "A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems," *European Journal of Operations Research*, vol. 232, no. 3, pp. 479–488, 2014.
[25] J. C. Hull, *Options, Futures and Other Derivatives*, 8th ed. Pearson, 2011.
[26] K. Asanovic, B. C. Catanzaro, D. A. Patterson, and K. A. Yelick, "The Landscape of Parallel Computing Research : A View from Berkeley," *EECS Department University of California Berkeley*, vol. 18, no. UCB/EECS-2006-183, p. 19, 2006.
[27] J. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science Engineering*, vol. 12, no. 3, pp. 66 –73, May-June 2010.
[28] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/
[29] T. Achterberg, "SCIP: Solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.
[30] T. Koch, "Rapid mathematical prototyping," Ph.D. dissertation, Technische Universität Berlin, 2004.
[31] C. de Schryver, M. Jung, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "Energy Efficient Acceleration and Evaluation of Financial Computations towards Real-Time Pricing," in *Knowledge-Based and Intelligent Information and Engineering Systems*, A. König, A. Dengel, K. Hinkelmann, K. Kise, R. Howlett, and L. Jain, Eds. Springer Berlin Heidelberg, 2011, vol. 6884, pp. 177–186.

Gordon Inggs received the B.Sc.(Eng.) *Hons.* and M.Sc.(Eng.) degrees in electrical engineering from the University of Cape Town, and recently completed his Ph.D. degree in Computer Engineering at Imperial College London. His research interests include distributed, heterogeneous computing, and domain specific programming tools.

David B. Thomas received the M.Eng. and Ph.D. degrees in computer science from Imperial College London. He is a Lecturer with the Electrical and Electronic Engineering Department, Imperial College London. His research interests include hardware-accelerated cluster computing, Monte Carlo simulation, random number generation, and financial computing.

Wayne Luk received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from the University of Oxford. He is a Professor of computer engineering with Imperial College London, and a Visiting Professor with Stanford University and Queens University Belfast. His research interests include theory and practice of customising computing for application domains, such as multimedia, networking, and finance.