Speedup and Power Scaling Models for Heterogeneous Many-Core Systems

Ashur Rafiev, Mohammed A. N. Al-hayanni, *Student member, IEEE,* Fei Xia, Rishad Shafik, *Member, IEEE,* Alexander Romanovsky, Alex Yakovlev, *Fellow, IEEE*

***Abstract*—Traditional speedup models, such as Amdahl's Law, Gustafson's, and Sun and Ni's models, have helped the research community and industry to better understand the performance capabilities of systems and the parallelizability of applications. Mostly targeting homogeneous hardware platforms or a limited form of processor heterogeneity, these models do not cover newly emerging multi-core heterogeneous architectures. This paper reports novel speedup and energy consumption models based on a more general representation of heterogeneity, called normal form heterogeneity, supporting a wide range of heterogeneous many-core architectures. The modelling method aims to predict system energy efficiency and performance ranges and facilitates research and development for the hardware and system software levels. Extensive experimentation on an off-the-shelf big.LITTLE heterogeneous platform validates the models showing less than 1% error for speedup and less than 4% error for power dissipation. The practical use of the method is demonstrated with a quantitative study of system load balancing efficiency.**

***Index Terms*—Heterogeneous systems, speedup modelling, enrgy-aware systems, load balancing, Amdahl's law, multi-core processors**

I. Introduction

FROM the early days of computing systems, there has been a persistent engineering effort to improve computation speed by distributing the work across multiple devices. Predicting the system's gain in performance, called the *speedup*, has been a major focus in this area of the research. Amdahl's law has been known since 1967 [1]. It assumes that a fixed workload is executed in n processors and compares the performance with the same workload executed in a single processor. The model shows that the speedup will quickly saturate with increasing n if the workload requires synchronization. In 1988, Gustafson introduced the principle of workload scaling pertaining to the fixed time model [2]. This model proposes to extend the workload proportionally to system's scalability with the result of having linear increase in the speedup. In 1990, Sun and Ni suggested a new model, which included extended workload calculations by considering the capability of the memory [3], [4].

Over the years, technology scaling has facilitated significant performance improvement at reduced power consumption through increased operating frequency and smaller device geometries [5]. The number of transistors per unit of area have increased substantially conforming to Moore's [6] and Koomey's laws [7], and Pollack's rule suggests that performance is increasing approximately proportional to the square root of the complexity [8].

As a result, nowadays almost every consumer device or embedded system uses the computational power of multi-core processing. The number of cores in a device is constantly growing, hence the speedup scaling models remain of high importance. The convenience of using Amdahl's law and derived models is in that they do not require complex modelling and simulation of individual inter-process communications. Instead, these models operate on the average platform and application characteristics and provide simple analytical solutions that project system's capabilities in a clear and understandable way. They provide a valuable insight into system scalability and have become pivotal for multi-scale systems research. However, it is still important to keep the models up to date, to make sure they stay relevant and correctly represent novel aspects of platform design.

From the increase in system complexity and integration, the concept of heterogeneous computation has emerged. Initially, the heterogeneity appeared in a form of specialized accelerators, like GPU and DSP. In recent years, multiple types of CPU cores in a single device have also been made popular. For instance, the ARM big.LITTLE processor has found a wide use in mobile devices [9]. Heterogeneous systems pose additional engineering and research challenges. In the area of scheduling and load balancing, the aim is to improve core utilization for more efficient use of the available performance. Operating systems traditionally implement symmetric multi-processor (SMP) scheduling algorithms designed for homogeneous systems, and ARM have done dedicated work on modifying the Linux kernel to make load balancing suitable for their big.LITTLE processor [10].

In addition to performance concerns, power dissipation management is also a significant issue in scalable systems: according to Dennard's CMOS scaling law [11] despite smaller geometries the power density of devices remains constant.

Hill and Marty extended Amdahl's speedup model to cover simple heterogeneous configurations consisting of a single big core and many smaller ones of exactly the same type [12], which relates to the CPU-GPU type of heterogeneity. The studies in [13], [14] extended Hill-Marty analysis to all three major speedup models. The problem of energy efficiency has been addressed in [15] for the homogeneous and simple heterogeneous Amdahl's model.

A. Rafiev, M. Al-hayanni, F. Xia, R. Shafik, A. Romanovsky, and A. Yakovlev are with Newcastle University, UK
E-mail: {ashur.rafiev, m.a.n.al-hayanni, fei.xia, rishad.shafik, alexander.romanovsky, alex.yakovlev}@ncl.ac.uk
M. Al-hayanni is also with University of Technology and HCED, Iraq
This work is supported by EPSRC/UK as a part of PRiME project EP/K034448/1.

TABLE I
EXISTING SPEEDUP MODELS AND THE PROPOSED MODEL

	homogeneity	heterogeneity	power	Amdahl's law	Gustafson's model	Sun and Ni's model
[1]	yes	no	no	yes	no	no
[2]	yes	no	no	yes	yes	no
[3]	yes	no	no	yes	yes	yes
[12]	yes	simple	no	yes	no	no
[15]	yes	simple	yes	yes	no	no
[13], [14]	yes	simple	no	yes	yes	yes
proposed models	yes	normal form (Section 3)	yes	yes	yes	yes

A. Research Contributions

In order to be relevant to more general and emerging types of heterogeneous systems, new types of models need to be developed. This paper extends the classical speedup models to a so-called normal form representation of heterogeneity, which describes core performances as a vector. This representation can fit a wider range of systems, including the big.LITTLE processor or homogeneous processors with multiple dynamic voltage-frequency scaling (DVFS) islands. The initial work on this topic has been published in [16], which includes the derivation of the speedup models as well as a set of power models for this extended representation of heterogeneity. In addition to fixed-workload Amdahl's law, workload scaling from the works of Gustafson and Sun-Ni have also been consirered. In the current publication we expand the work by addressing the effects of workload distribution and load balancing, and also explore additional modes of workload scaling relevant only to heterogeneous systems. We discover that the presented models inherit certain limitations from the Amdahl's law, which may be significant for heterogeneous modelling and need to be taken into account. The paper provides a concise discussion on the matter. In addition, an extensive set of experiments has been carried out to validate the proposed models and to explore their practical use.

This paper makes the following contributions:

- extending the classical speedup models to normal form heterogeneity in order to represent modern examples of heterogeneous systems;
- extending heterogeneous speedup models to include power and energy estimation and studying the performance-energy trade-offs in the heterogeneous systems;
- clarifying the limitations of the Amdahl-like heterogeneous models and outlining further challenges of heterogeneous speedup and power modelling;
- validating the models on a real heterogeneous platform under a set of carefully controlled model parameters;
- practically using the models to evaluate the efficiency of the Linux scheduler's load balancing while running realistic workload in a heterogeneous system.

Table 1 compares this paper's contributions to the range of related research publications.

The experimental work presented in this paper has been carried out on the Odroid-XU3 [17] development platform centred around ARM big.LITTLE Cortex A7-A15 cores.

The paper is organized as follows. Section 2 gives an overview of the existing homogeneous and heterogeneous speedup models. Section 3 discusses the model assumptions and formally defines the normal form heterogeneous system's structure. Sections 4 and 5 present the new heterogeneous speedup and power models respectively. Section 6 experimentally validates the models. Section 7 shows the experiments with real life benchmarks. Section 8 concludes the work.

II. EXISTING SPEEDUP MODELS

In homogeneous systems all cores are identical in terms of performance, power, and workload execution.

For a homogeneous system we consider a system consisting of n cores, each core having a performance of $\theta = \frac{I}{t(1)}$, where I is the given workload and $t(1)$ is the time needed to execute the workload on the core. This section describes various existing models for determining the system's speedup $S(n)$ in relation to a single core, which can be used to find the performance $\Theta(n)$ of the system:

$$\Theta(n) = \theta S(n). \tag{1}$$

Amdahl-like speedup models are built around the parallelizability factor p, $0 \leq p \leq 1$, which reflects the application's capability of performing parallel computation. Given a total workload of I, the parallel part of a workload is pI and the sequential part is $(1-p)I$.

A. Amdahl's Law (Fixed Workload)

The general idea of this model is to compare execution time for some fixed workload I on a single core with the execution time for the same workload on the entire n-core system [1].

Time to execute workload I on a single core is $t(1)$, whereas $t(n)$ adds up the sequential execution time on one core at the performance θ and the parallel execution time on all n cores at the performance $n\theta$:

$$t(1) = \frac{I}{\theta}, \quad t(n) = \frac{(1-p)I}{\theta} + \frac{pI}{n\theta}, \tag{2}$$

thus the speed up can be found as follows:

$$S(n) = \frac{t(1)}{t(n)} = \frac{1}{(1-p) + \frac{p}{n}}. \tag{3}$$

B. Gustafson's Model (Fixed Time)

Gustafson re-evaluated the fixed workload speedup model to derive a new fixed time model [2]. In this model, the workload increases with the number of cores, while the execution time is fixed. An important note is that the workload scales asymmetrically: the parallel part is scaled to the number of cores, whilst the sequential part is not increased.

Let's denote the initial workload as I and extended workload as I'. The time to execute initial workload and extended

workload are $t(n)$ and $t'(n)$ respectively. The workload scaling ratio can be found from:

$$t(1) = \frac{I}{\theta}, \quad t(n) = \frac{(1-p)I}{\theta} + \frac{pI'}{n\theta}, \tag{4}$$

and, since $t(1) = t(n)$, the extended workload can be found as $I' = nI$. The time that would take to execute I' on a single core is:

$$t'(1) = \frac{(1-p)I}{\theta} + \frac{pnI}{\theta}, \tag{5}$$

which means that the achieved speedup equals to:

$$S(n) = \frac{t'(1)}{t(1)} = (1-p) + pn. \tag{6}$$

Not all applications can provide workload extension only in the parallelizable part without changing the sequential part, hence there is a limitation on the applicability of the extended workload models. A typical example is an application that changes the computation quality depending on the number of available cores. The main contribution of Gustafson's model, however, is to show that it is possible to build an application that scales to multiple cores without suffering saturation.

C. Sun and Ni's Model (Memory Bounded)

Sun and Ni took into account the previous two speedup models by considering the memory bounded constraints [3], [4]. In this model the execution time and the workload change according to the memory capability. The parameter $g(n)$ reflects the scaling of the workload in relation to scaling the memory with the number of cores:

$$I' = g(n)\, I. \tag{7}$$

A typical example $g(n)$ is given for an $m \times m$ matrix multiplication, which has the memory requirement of $O\left(m^2\right)$ and the computation cost (workload) of $O\left(m^3\right)$. In this case, $g(n) = n^{\frac{3}{2}}$.

The model calculates the speedup as follows:

$$S(n) = \frac{t'(1)}{t'(n)} = \frac{(1-p) + pg(n)}{(1-p) + \frac{pg(n)}{n}}. \tag{8}$$

Because the workload is scaled by $g(n)$ according to (7), one of the important properties of this model is that for $g(n) = 1$ Sun and Ni's model (8) transforms into Amdahl's law (3), and for $g(n) = n$ it becomes Gustafson's law (6). Further in this paper, we do not specifically relate $g(n)$ to the memory access or any other property of the system and consider it as a given or determined parameter pertaining to a general case of workload scaling.

D. Existing Heterogeneous Models

Previous attempts to extend speedup laws to heterogeneous systems were mainly focused on a single high performance core and many smaller cores of the same type [12].

Core performances are related to some base-core equivalent (BCE), which is considered to have $\theta = 1$. This model studies a system with one big core having $\Theta(r)$ relative performance and $(n-r)$ little cores with BCE performances, as shown in Figure 1(b). The sequential workload is executed on the faster core, while the parallel part exercises all cores simultaneously. This transforms Amdahl's law (3) as follows:

$$S(n, r) = \frac{1}{\frac{(1-p)}{\Theta(r)} + \frac{p}{\Theta(r)+(n-r)}}. \tag{9}$$

In this work we aim to cover more diverse cases of heterogeneity pertaining to such modern architectures as ARM big.LITTLE [17] which are not directly covered by existing speedup models.

III. Heterogeneous Platform Assumptions

Homogeneous models are used to compare the speedup between different numbers of cores. Similarly, heterogeneous models should compare the speedup between core configurations, where each configuration defines the number of cores in each available core type. This section discusses the problems of modelling consistency across different core types and provides the foundation for all heterogeneous models presented later in this paper.

A. The Challenges of Heterogeneous Modelling

Heterogeneous models must capture the performance and other characteristics across different types of cores in a comparable way. Such a comparison is not always straightforward, and in many ways similar to cross-platform comparison. This section discusses the assumptions behind Amdahl's law and similar models under the scope of heterogeneous modelling and outlines the limitations they may cause.

1) Hardware-dependent parallelizability: In the models presented in Section 2, there is a clear time-separation of the synchronous and parallel executions of the entire workload. In other words, all threads synchronize at the same time. These models do not explore complex interactions between the processes, hence they do not provide exact timing predictions and should not be used for time-critical analyses like real-time systems research. Solving for process interactions is possible with Petri Net simulations [18] or process algebra [19]. Amdahl-like models, in contrast, focus on generic analytical solutions that give approximate envelopes for platform capabilities.

These models use parallelizability factor p as one of the main parameters to the model, however they give little explanation on how to obtain p. This problem has been a challenge for numerous research efforts [20]. A nave intuition is that p is a property of an algorithm. However, there is a number of known embarrassingly-parallelizable algorithms ($p = 1$), but it has never been possible to achieve this level of parallelization running such algorithms on real platforms.

The real value of p is a combined property of an algorithm running on a specific hardware, and represents algorithm parallelizability p_a being modulated by platform architecture's parallelizability p_h. In a simplified case, there are two sources of sequential execution: computation synchronization $(1-p_a)\, I$, required by the algorithm itself, and the hardware critical section $(1-p_h)\, p_a I$, when algorithmically parallel computations have to wait for shared hardware resources. Hence, the

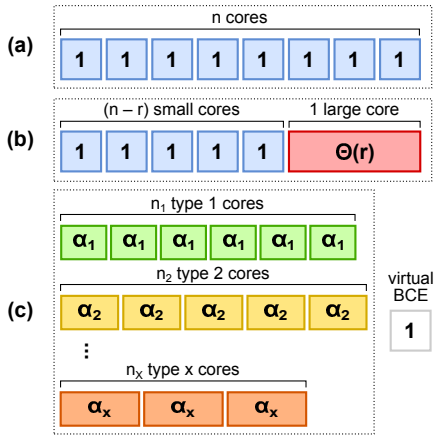

Fig. 1. The proposed extended structure of a heterogeneous system (c) compared to a homogeneous system (a) and the previous assumption [12] on heterogeneity (b). The numbers in the core boxes denote the equivalent number of BCEs.

combined sequential workload is $(1 - p_a p_h) I$ and the fully parallel part is $p_a p_h I$, which leads to:

$$p = p_a p_h. \tag{10}$$

On the surface, this equation divides p into application-dependent and hardware-dependent components, but this is not quite true as p_h is also dependent on the instructions being executed. As an example, consider two embarrassingly-parallelizable ($p_a = 1$) applications: one performing big calculation on a small set of data, and another is performing small calculation over a large amount of data. The second application requires considerably more memory accesses per unit of output, and if the memory is a shared resource, this would cause different p_h for the two algorithms on the same processor. Hence, (10) does not solve the problem, but pushes it one step further.

From the standpoint of heterogeneous modelling, the potential differences in p_h between core types or cache islands will cause the overall p to change between core configurations. In this paper, we do not attempt to solve this challenge. As demonstrated further in this paper, it is still possible to build heterogeneous models around a constant p and use a range of possible values $[p_{\min}, p_{\max}]$ to determine the system's minimum and maximum speedup capabilities.

2) Workload equivalence and performance comparison: Workload is a model parameter that links performance with the execution time. In many cases, a popular metric for performance is instructions per second (IPS), in which case a workload is characterized by its number of instructions. IPS is convenient as it is an application-independent property of the platform; it is also used for deriving power optimization metrics such as energy per instruction.

In heterogeneous models, it is important to have a consistent metric across all core types. For devices of different architecture types, the same computation may be compiled into different numbers of instructions. In this case, the total number of instructions can no longer meaningfully represent the same workload, and IPS cannot be universally used for cross-platform performance comparison. This is particularly clear when comparing CPU and GPU devices.

In order to build a valid cross-platform performance comparison model, we need to reason about the workload as a meaningful computation, and two workloads are considered equivalent as long as they perform the same task. In this paper we measure workload in so-called "workload items", which can be defined on a case by case basis depending on the practical application. Respectively, instead of energy per instruction, we use energy per workload item.

Hill and Marty's model, presented in Section 2.4, describes the performance difference between the core types as $\Theta(r)$. In real life, this relation is application dependent, as will be demonstrated in Section 6. Differences in hardware, such as pipeline depth and cache sizes, cause performance differences on a per-instruction basis [21]. As a result, even within the same instruction set, core type i may execute workload A faster than core type j, but core type j may execute workload B faster than core type i. Hence, A and B must use different performance ratios to describe the same heterogeneous platform.

B. Platform Assumptions

We build our models under the assumptions listed below. These assumptions put limitations on the models as discussed earlier in this section. However, the same assumptions are used in the classical Amdahl's law and similar models, hence they do not reduce the applicability of the presented models.

- The models and model parameters are both application and hardware specific.
- The relation between performances of cores of different types can be approximated to a constant ratio.
- The parallelizability factor p can be approximated by a constant and is known or can be determined (exactly or within a range).
- Environmental factors, such as temperature, are not considered.

Inter-core communication overheads, addressed in the Li and Malek's model [22], are not considered in this paper and are the subject of future work.

C. Normal Form Representation of Heterogeneity

Performance-wise, the models presented in subsequent sections describe heterogeneity using the following normal form representation.

The normal form of heterogeneous system configuration considered in this paper consists of x clusters (types) of homogeneous cores with the numbers of cores defined as a vector $\overline{n} = (n_1, \ldots, n_x)$. The total number of cores in the system is denoted as $N = \sum_{i=1}^{x} n_i$. Vector $\overline{\alpha} = (\alpha_1, \ldots, \alpha_x)$ defines the performance of each core by cluster (type) in relation to some base core equivalent (BCE), such that for all $1 \leq i \leq x$ we have $\theta_i = \alpha_i \theta$. As discussed earlier, the parameter $\overline{\alpha}$ is application- and platform-dependent. The structure is shown in Figure 1(c).

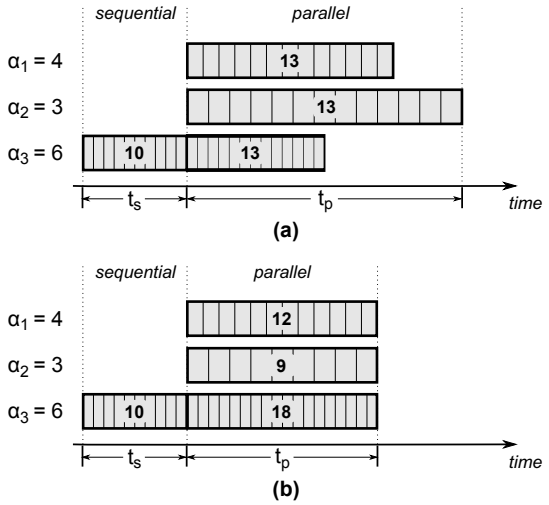

Fig. 2. Workload distribution examples following (a) equal-share model and (b) balanced model.

IV. Proposed Speedup Models

This section extends homogeneous speedup models for determining the speedup $S(\overline{n})$ of a heterogeneous system in relation to a single BCE, which can then be used to find the performance of the system using (1).

A. Workload Distribution Models

The fundamental idea behind the speedup modelling is that the cores do not contribute to overall system performance when idle. Homogeneous models distinguish two states of performance: the parallel execution exercises all cores, and the sequential execution exercises only one core while others are idle. The cores in such systems are considered identical, hence they all execute equal shares of the parallelizable part of the workload and finish at the same time. As the result, the combined performance of the cores working in parallel is θn. In heterogeneous systems this is not as straightforward: each type of cores works at a different performance rate, hence the execution time depends greatly on the workload distribution between the cores. Imperfect distribution causes some cores to finish early and become idle, even when the parallelizable part of the workload has not been completed.

In real systems, the scheduler is assisted by a load balancer, whose task is to redistribute the workload during run-time from busy cores to idle cores, however its efficiency is not guaranteed to be optimal [23]. The actual algorithm behind the load balancer may vary between different operating systems, and the load balancer typically has access to run-time only information like CPU time of individual processes and the sizes of waiting queues. Hence it is virtually impossible to accurately describe the behaviour of the load balancer as an analytical formula. In this section we address the problem by studying two boundary cases, which may provide a range of minimum and maximum parallel performances.

By definition, the *total execution time* for the workload I is a sum of sequential and parallel execution times, $t_s(\overline{n})$ and $t_p(\overline{n})$, and it represents the time interval between the first instruction in I starting and the last instruction in I finishing, meaning that, during a parallel execution, only the longest running core has an effect on the total execution time. In other words:

$$t_p(\overline{n}) = \max_{i=1}^{x} \frac{I_{pi}}{\alpha_i \theta}, \tag{11}$$

where I_{pi} is a share of the parallelizable workload (pI) for a single core of the type i.

To be analogous to the homogeneous models and to simplify our equations, we also define the system's parallel performance via the *performance-equivalent number* of BCEs denoted as N_α:

$$N_\alpha = \frac{pI}{t_p(\overline{n})\,\theta} = pI \cdot \min_{i=1}^{x} \frac{\alpha_i}{I_{pi}}. \tag{12}$$

1) Equal-share workload distribution: In homogeneous systems, the parallelizable workload is equally split between all cores. As a results, many legacy applications, developed with the homogeneous system architecture in mind, would also equally split the workload by the total number of cores (threads): $I_{pi} = \frac{pI}{N}$, which leads to a very inefficient execution in heterogeneous systems, where everyone is waiting for the slowest core (thread), as illustrated in Figure 2(a). In this case, N_α is calculated from the minimum of $\overline{\alpha}$:

$$N_\alpha = N \cdot \min_{i=1}^{x} \alpha_i. \tag{13}$$

The above equation implies that the workload cannot be moved between the cores. If the system load balancer is allowed to re-distribute the work, then the real N_α may be greater than (13). This equation can be used to define a lower performance bound corresponding to nave scheduling policy with no balancing.

2) Balanced workload distribution: Figure 2(b) shows the ideal case of workload balancing, which implies zero waiting time, hence all cores should theoretically finish at the same time. In other words, a parallel execution time $t_{pi}(\overline{n})$ in core type i must be equal to $t_p(\overline{n})$ for all $1 \le i \le x$:

$$t_p(\overline{n}) = t_{pi}(\overline{n}) = \frac{I_{pi}}{\alpha_i \theta}. \tag{14}$$

Because of this, we can agree that, from (12), individual core workload I_{pi} can be found for all $1 \le i \le x$ as follows:

$$I_{pi} = \frac{\alpha_i pI}{N_\alpha}. \tag{15}$$

We also know that all individual core workloads must add up to the total parallelizable workload:

$$pI = \sum_{i=1}^{x} n_i I_{pi}, \tag{16}$$

and we can solve equations (15) and (16), giving us N_α for optimal workload distribution:

$$N_\alpha = \sum_{i=1}^{x} \alpha_i n_i. \tag{17}$$

$N_\alpha \theta$ represents the system's performance during the parallel execution, hence N_α values from (13) and (17) define the range for heterogeneous system parallel performances. A load balancer that violates the lower bound (13) is deemed to be worse than nave. The upper bound (17) represents the theoretical maximum, and it cannot be exceeded.

B. Heterogeneous Amdahl's Law

We assume that the sequential part is executed on a single core in the cluster s, hence the system's performance during sequential execution is $\alpha_s\theta$. In Section 4.1 we defined parallel performance as $N_\alpha\theta$. Hence, the time to execute the fixed workload I on the given heterogeneous system is:

$$t(\overline{n}) = t_s(\overline{n}) + t_p(\overline{n}) = \frac{(1-p)I}{\alpha_s\theta} + \frac{pI}{N_\alpha\theta}. \tag{18}$$

The speedup in relation to a single BCE is:

$$S(\overline{n}) = \frac{t(1)}{t(\overline{n})} = \frac{1}{\frac{(1-p)}{\alpha_s} + \frac{p}{N_\alpha}}. \tag{19}$$

One can verify that this equation also covers Hill-Marty's model (9), in which case $\overline{n} = (n-r, 1)$, $\overline{\alpha} = (1, \Theta(r))$, $\alpha_s = \Theta(r)$, and N_α is calculated for the balanced workload distribution (17).

C. Workload Scaling

Like in the homogeneous case, Amdahl's law works with a fixed workload, while Gustafson and Sun-Ni allow changing the workload with respect to the system's capabilities. In this section we consider a general assumption on workload scaling, which defines the extended workload using characteristic functions $g(\overline{n})$ and $h(\overline{n})$ as follows:

$$I' = h(\overline{n}) \cdot ((1-p)I + pg(\overline{n})I), \tag{20}$$

where $h(\overline{n})$ represents the symmetric scaling of the entire workload, and $g(\overline{n})$ represents the scaling of the parallelizable part only.

The sequential and parallel execution times are respectively:

$$t'_s(\overline{n}) = \frac{(1-p)I}{\alpha_s\theta}, \quad t'_p(\overline{n}) = \frac{pg(\overline{n})I}{N_\alpha\theta}. \tag{21}$$

Hence, in the general case, for given workload scaling functions $g(\overline{n})$ and $h(\overline{n})$, the speedup is calculated as follows:

$$S(\overline{n}) = \frac{I'}{\left(t'_s + t'_p\right)\theta} = \frac{(1-p) + pg(\overline{n})}{\frac{(1-p)}{\alpha_s} + \frac{pg(\overline{n})}{N_\alpha}}. \tag{22}$$

Note that the speedup does not depend on the symmetric scaling $h(\overline{n})$. Indeed, the execution time proportionally increases with the workload, and the performance ratio (i.e. the speedup) remains constant. However, changing the execution time is important for the fixed-time Gustafson's model.

D. Heterogeneous Gustafson's Model

In the Gustafson model, the workload is extended to achieve equal time execution: $t'(\overline{n}) = t(1)$. For homogeneous Gustafson's model: $g(\overline{n}) = n$ and $h(\overline{n}) = 1$. For a heterogeneous system, there are more than one way to achieve equal time execution.

1) Purely parallel scaling mode: The maximum speedup for equal time execution is achieved by scaling only the parallel part, i.e. $h(\overline{n}) = 1$. We know that Gustafson's model requires equal execution time, hence:

$$t'_s(\overline{n}) + t'_p(\overline{n}) = t(1), \tag{23}$$

which leads to:

$$\frac{pg(\overline{n})}{N_\alpha} = 1 - \frac{(1-p)}{\alpha_s}. \tag{24}$$

From this, we can find that:

$$g(\overline{n}) = \left(1 - \frac{(1-p)}{\alpha_s}\right)\frac{N_\alpha}{p}, \tag{25}$$

however this equation puts a number of restrictions on the system. Firstly, it doesn't work for $p = 0$, because it is not possible to achieve equal time execution for a purely sequential program if $\alpha_s \neq 1$ and only the parallel workload scaling is allowed. Secondly, a negative $g(\overline{n})$ does not make sense, hence the relation $\alpha_s > (1-p)$ must hold true. This means that the sequential core performance must be high enough to overcome the lack of parallelization. Another drawback of this mode is that it requires the knowledge of p in order to properly scale the workload.

In this scenario, the speedup is calculated as:

$$S(\overline{n}) = (1-p) + \left(1 - \frac{(1-p)}{\alpha_s}\right)N_\alpha. \tag{26}$$

2) Classical scaling mode: In order to remove the restrictions of the purely parallel scaling mode, and to provide a model generalizable to $p = 0$, we need to allow scaling of the sequential execution. However, since this mode potentially increases the sequential execution time, it exercises the cores less efficiently than the previous mode and leads to lower speedup. In this case, (23) can be updated to:

$$h(\overline{n}) \cdot \left(t'_s(\overline{n}) + t'_p(\overline{n})\right) = t(1). \tag{27}$$

From this, in the case of $p = 0$, we find that $h(\overline{n}) = \alpha_s$. And for the case of $p > 0$ and $h(\overline{n}) = \alpha_s$:

$$g(\overline{n}) = \left(\frac{1}{h(\overline{n})} - \frac{(1-p)}{\alpha_s}\right)\frac{N_\alpha}{p} = \frac{N_\alpha}{\alpha_s}. \tag{28}$$

This scaling mode relates to the classical homogeneous Gustafson's model, which requires $g(n)$ to be proportional to the ratio between the system performances of the parallel and sequential executions. In the homogeneous case, if the sequential performance is θ, the parallel performance would be $n\theta$, leading to $g(n) = n$.

For the heterogeneous Gustafson's model in classical scaling mode, the speedup is calculates as:

$$S(\overline{n}) = \alpha_s(1-p) + pN_\alpha. \tag{29}$$

V. Proposed Power and Energy Models

We base our power models on the concept of power state modelling, in which a device has a number of distinct power states, and the average power over an execution is calculated from the time the system spent in each state.

For each core in the system we consider two power states: active and idle. Lower power states like sleeping and shutting down the cores are not included in the presented models, however it is possible to extend the models to cater to these effects. Let's denote the active power of a core in a homogeneous system as w_a and the idle power of a core as w_0. Active power can also be expressed as a sum of idle power w_0 and effective power w that is spent on workload computation, $w_a = w_0 + w$. In this view, the idle component is no longer dependent on the system's activity and can be expressed as a system-wide constant term W_0, called *background power.* The total power dissipation of the system is:

$$W_{total} = W_0 + W(\overline{n}), \tag{30}$$

$W(\overline{n})$ is the total effective power of active cores – this is the focus of our models. The constant term of background power W_0 can be studied separately.

A. Power Modelling Basics

In the normal from representation of a heterogeneous system (Section 3), the difference between power dissipations of the cores is expressed by the vector $\overline{\beta} = (\beta_1, \ldots, \beta_x)$, which defines the effective power in relation to a BCE's effective power, such that for all $1 \leq i \leq x$ we have effective power $w_i = \beta_i w$.

The effective power model can be found as a time-weighted average of the sequential effective power w_s and parallel effective power w_p of the system:

$$W(\overline{n}) = \frac{w_s t'_s(n) + w_p t'_p(n)}{t'_s(n) + t'_p(n)}, \tag{31}$$

where $t'_s(n)$ and $t'_p(n)$ are the speedup-dependent times required to execute sequential and parallel parts of the extended workload respectively.

In a homogeneous system:

$$w_s = w,\ w_p = nw. \tag{32}$$

In a heterogeneous system, if we execute the sequential code on a single core s:

$$\begin{aligned} w_s &= \beta_s w, \\ w_p &= N_\beta w, \end{aligned} \tag{33}$$

which gives for the balanced case of parallel execution (17):

$$N_\beta = \sum_{i=1}^{x} \beta_i n_i \tag{34}$$

For equal-share execution (13), N_β is calculated as follows:

$$N_\beta = \min \overline{\alpha} \cdot \sum_{i=1}^{x} \frac{\beta_i n_i}{\alpha_i}. \tag{35}$$

N_β is called a *power-equivalent number* of BCEs. Heterogeneous power models will transform into homogeneous if $\alpha_s = \beta_s = 1$ and $N_\alpha = N_\beta = n$.

B. Power Distribution and Scaling Models

We express the scaling of effective power in the system via the speedup and the *power distribution* characteristic function $D_w(\overline{n})$:

$$W(\overline{n}) = w D_w(\overline{n}) S(\overline{n}). \tag{36}$$

$D_w(\overline{n})$ represents the relation between the power and performance in a heterogeneous configuration. Since the speedup models are known from Section 4, this section focuses on finding the matching power distribution functions.

From (33) and (31), we can find that in the general case:

$$D_w(\overline{n}) = \left(\beta_s t'_s(n) + N_\beta t'_p(n)\right) \cdot \frac{\theta}{I'}, \tag{37}$$

thus substituting the workload scaling definition (20) and execution times (21) will give us:

$$D_w(\overline{n}) = \frac{\frac{\beta_s}{\alpha_s}(1-p) + pg(\overline{n})\frac{N_\beta}{N_\alpha}}{(1-p) + pg(\overline{n})}. \tag{38}$$

It is worth noting that for homogeneous systems, $D_w(n) = 1$ in all cases, and the effective power equation will transform into:

$$W(n) = wS(n), \tag{39}$$

i.e. in homogeneous systems the power scales in proportion to the speedup.

Power distribution for Amdahl's workload: For Amdahl's workload, $g(\overline{n}) = 1$, hence the power distribution function becomes:

$$D_w(\overline{n}) = \frac{\beta_s}{\alpha_s}(1-p) + p \cdot \frac{N_\beta}{N_\alpha} \tag{40}$$

Power distribution for Gustafson's workload: Following the same general form (38) for the effective power equation, we can find power distribution functions $D_w(\overline{n})$ for two cases of workload scaling described in Section 4.4.

For the classical scaling mode:

$$D_w(\overline{n}) = \frac{\beta_s(1-p) + pN_\beta}{\alpha_s(1-p) + pN_\alpha}. \tag{41}$$

For the purely parallel scaling mode:

$$D_w(\overline{n}) = \frac{\beta_s(1-p) + (\alpha_s - (1-p))N_\beta}{\alpha_s(1-p) + (\alpha_s - (1-p))N_\alpha}. \tag{42}$$

C. Energy and Power-Normalized Performance

Power modelling is typically used for optimizing system power dissipation. Due to the power-performance trade-off, advanced metrics are required as optimization targets. For example, power-normalized performance (performance per Watt) represents the performance achievable at a given power capacity. This parameter is the reciprocal of energy per instruction (in our case, per workload item) E_i, which can be found from dividing the total power (30) by the system's performance (1):

$$E_i = \frac{W_{total}}{\Theta(n)} = \frac{W_0 + W(\overline{n})}{\theta S(\overline{n})}. \tag{43}$$

For a single BCE we can denote energy per workload item as a sum of effective energy e and idle energy. Applying the power model (36) to (43):

$$E_i = e \cdot D_w(\overline{n}) + \frac{E_0}{S(\overline{n})}. \tag{44}$$

TABLE II
CHARACTERIZATION EXPERIMENTS: SINGLE CORE EXECUTION

benchmark	sqrt		int		log	
base workload, iterations	$2.4 \cdot 10^8$		$4.08 \cdot 10^9$		$2.4 \cdot 10^8$	
core type i	A7	A15	A7	A15	A7	A15
measured execution time, ms	74943	79798	79254	63920	62857	35668
measured active power, W	0.2805	0.8402	0.2887	0.8400	0.3143	0.9474
power measurement std dev	0.61%	0.24%	0.55%	0.19%	0.55%	1.89%
calculated effective power, W	0.1234	0.4850	0.1316	0.4848	0.1572	0.5922
α_i	1	0.9392	1	1.2399	1	1.7623
β_i	1	3.9303	1	3.6839	1	3.7672

The system-wide sum of idle energy per workload item is denoted as E_0. This equation shows that the power distribution function $D_w(\overline{n})$ increases the effective component of the energy as more power-hungry cores are being active, and the speedup $S(\overline{n})$ decreases the idle energy component due to better core utilization. The total energy consumption during the execution of the extended workload I' is $E = E_i I'$.

Energy-delay product (EDP) is another optimization metric that improves energy and performance at the same time:

$$Et(\overline{n}) = W_{total} \cdot \left(\frac{I'}{\Theta(n)} \right)^2 . \quad (45)$$

VI. EXPERIMENTAL VALIDATIONS

This section validates the models presented in Sections 4 and 5 against the a of experiments on a real heterogeneous platform. In these experiments, the goal is to determine the accuracy of the models when all model parameters, such as parallelization factor p, are under control.

A. Platform Description

This study is based on a multi-core mobile platform, the Odroid-XU3 board [17]. The main part of it is the 28nm application processor Exynos 5422. It is an SoC hosting an ARM big.LITTLE heterogeneous processor consisting of four Cortex A7 cores (C0 to C3) and four Cortex A15 cores (C4 to C7). The big Cortex A15 is a high performance 32-bit core having 32KB instruction and 32KB data L1 caches and 2MB L2 cache and the maximum frequency of 2.0GHz. The LITTLE Cortex A7 is a low power 32-bit core including the same L1 cache size and 512 KB L2 cache, and the maximum frequency of 1.4 GHz. There are compatible Linux and Android distributions available for Odroid-XU3; in our experiments we used Ubuntu 14.04. This SoC also has four power domains: A7 power domain, A15 power domain, GPU and memory power domains. The Odroid-XU3 board allows per-domain DVFS using predefined voltage-frequency pairs.

The previous assumption by Hill and Marty for heterogeneous architectures, shown in Figure 1(b), cannot describe systems such as big.LITTLE. Our models do not suffer from these restrictions and can be applied to big.LITTLE and similar structures.

B. Benchmark Description and Model Characterization

The models operate on application- and platform-dependent parameters, which are typically unknown and imply high efforts in characterization. However, in order to prove that the proposed models work, it is sufficient to show that, if $\overline{\alpha}$, $\overline{\beta}$ and p are defined, the performance and power behaviour of the system follows the model's prediction. These parameters can be fixed by a synthetic benchmark. This benchmark does not represent realistic application behaviour and was designed only for validation purposes. Experiments with real application examples are presented in Section 7.

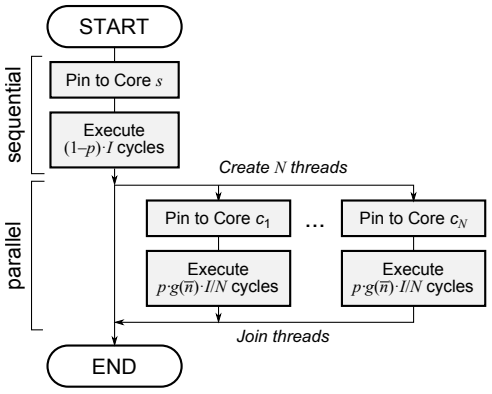

Fig. 3. Synthetic application with controllable parallelization factor and equal-share workload distribution. Parameter p, workload size I and scaling $g(\overline{n})$, the number of threads (cores) N, and the core allocation $s, \overline{c} = (c_1, \ldots, c_N)$ are specified as the program arguments.

The model characterization is derived from single core experiments. These characterized models are used to predict multi-core execution in different core configurations. The predictions are then cross-validated against experimental results.

1) Controlled parameters: The benchmark has been developed specifically for these experiments in order to provide control over the parallelization parameter p. Hence, p is not a measured parameter, but a control parameter that tells the application the ratio between the parallel (multi-threaded) and sequential (single thread) execution.

The application is based on POSIX threads, and its flow is shown in Figure 3. Core configurations, including homogeneous and heterogeneous, can be specified per application run as the sequential execution core s and the set of core allocations $\overline{c} = (c_1, \ldots, c_N)$, where N is the number of parallel threads; $s, c_j \in \{\mathrm{C1}, \ldots, \mathrm{C7}\}$ for $1 \leq j \leq N$. C0 is reserved for OS and power monitors. These variables define $\overline{n}$ used in the models. We do not shut down the cores and use per-thread core pinning via `pthread_attr_setaffinity_np` to avoid unexpected task migration.

The workload size I and the workload scaling $g(\overline{n})$ are

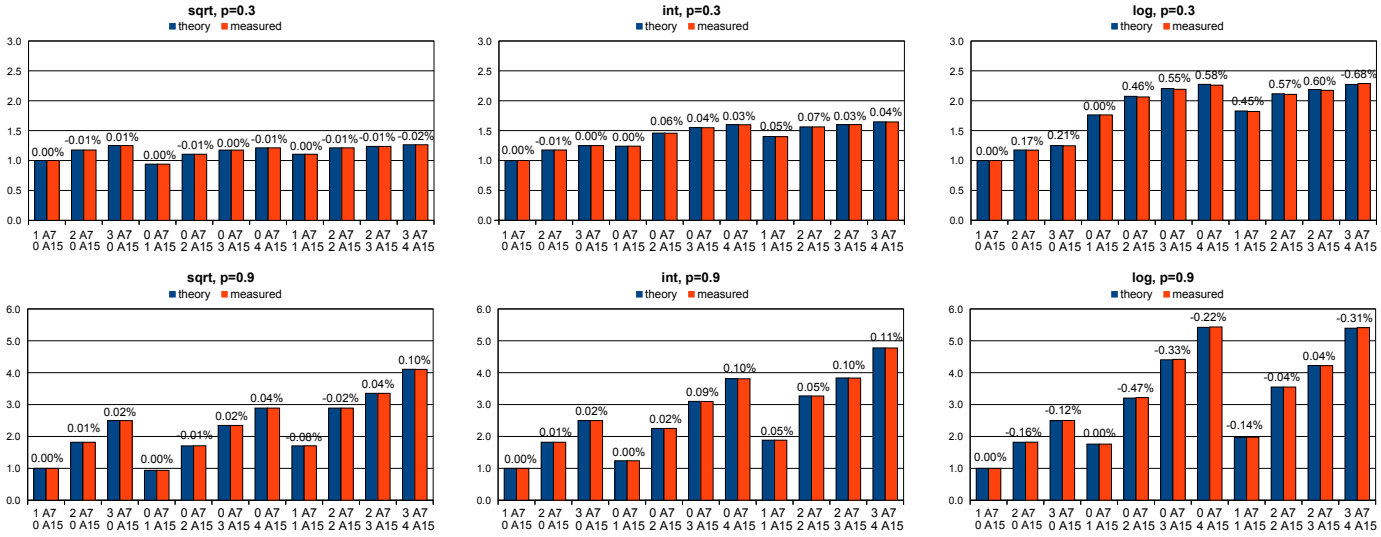

Fig. 4. Speedup validation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured speedup.

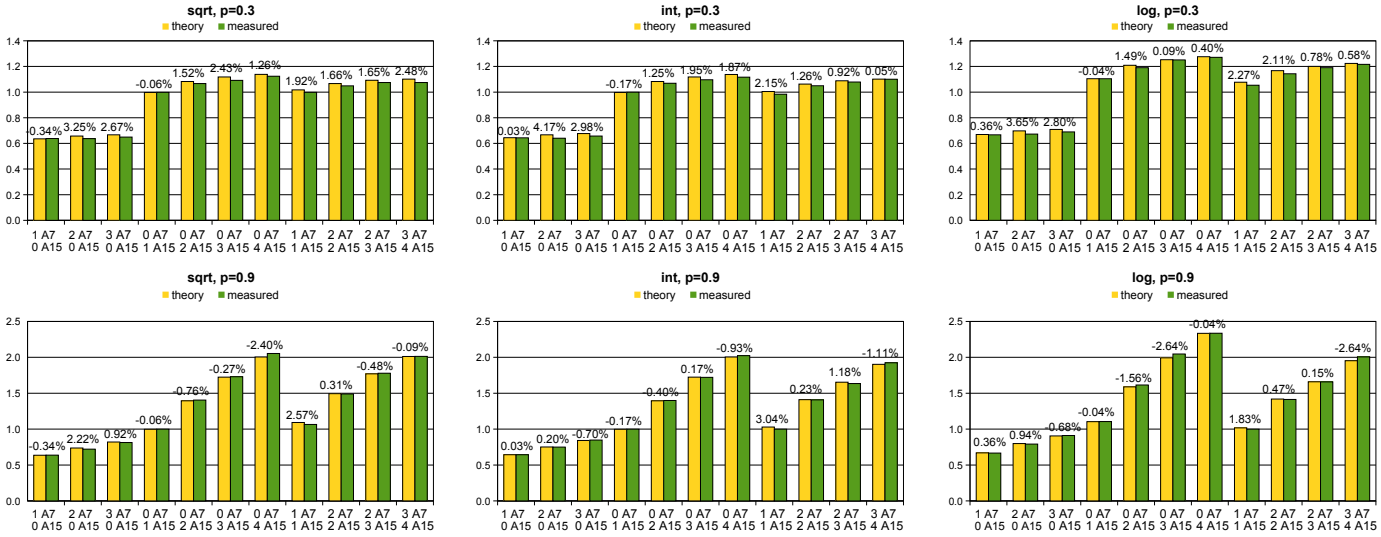

Fig. 5. Total power dissipation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured power.

also given parameters, which are used to test Gustafson's models against the Amdahl's law. The application implements three workload functions: square root calculation (sqrt), integer arithmetic (int), and logarithm calculation (log) repeated in a loop. These computation-heavy tasks use minimal memory access to reduce the impact of hardware on the controlled p. A fixed number of loop iterations represents one workload item. The functions are expected to give different performance characteristics, hence the characterization and cross-validation experiments are done separately for each function.

Figure 3 shows equal-share workload distribution, where each parallel thread receives equal number of $pg\left(\overline{n}\right)\frac{I}{N}$ workload items. This execution gives N_α and N_β that correspond to nave load balancing according to (13) and (35). Additionally, after collecting the characterization data for $\overline{\alpha}$, we implemented a version that uses $\overline{\alpha}$ to do optimal (balanced) workload distribution by giving each core $c_j \in \overline{c}$ a performance-adjusted workload of $pg\left(\overline{n}\right)\frac{I}{N}\cdot\frac{\alpha_j}{A}$, where $A=\sum_{j=1}^{N}\alpha_j$. This execution follows different N_α and N_β, which can be calculated from (17) and (34).

2) Relative performances of cores: All experiments in this section are run with both A7 and A15 cores at 1.4GHz. Running both cores at the same frequency exposes the effects of architectural differences on the performance. In addition, by avoiding higher frequencies we reduce the temperature effects and avoid throttling. In this study, we set BCE to A7, hence $\alpha_{\mathrm{A7}}=1$; and α_{A15} can be found as a ratio of single core execution times $\alpha_{\mathrm{A15}}=t_{\mathrm{A7}}\left(1\right)/t_{\mathrm{A15}}\left(1\right)$, as shown in Table 2. The three different functions provide different α_{A15} values.

It can be seen that A15 is unsurprisingly faster than A7 for integer arithmetic and logarithm calculation, however the square root calculation is faster on A7. This is confirmed multiple times in many experiments. We did not fully investigate the reason of this behaviour since the board's production and support have been discontinued, and this is in any case outside the scope of this paper. A newer version of the board, Odroid-

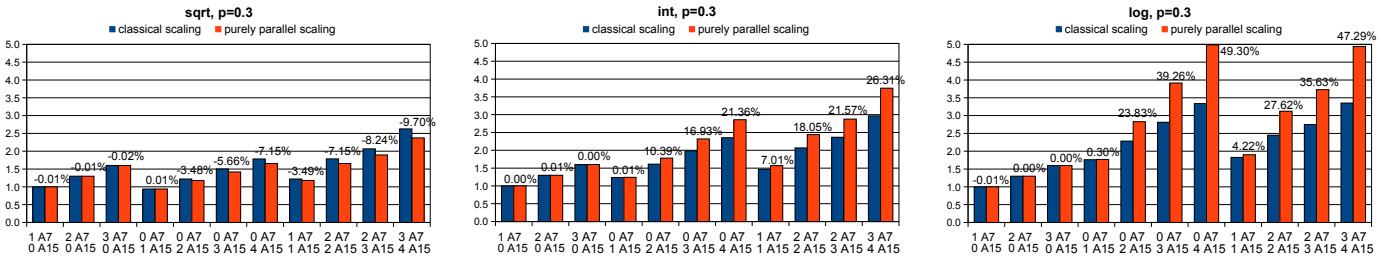

Fig. 6. Gustafson's model outcomes showing the the measured speedup gain from using the purely parallel workload scaling compared to the classical scaling; sqrt shows poor speedup because of $\alpha_s < 1$.

TABLE III
GUSTAFSON'S WORKLOAD SCALING CALCULATIONS

bench	n_{A7}	n_{A15}	classical $g(\overline{n})$	purely parallel $g(\overline{n})$	
				$p = 0.3$	$p = 0.9$
sqrt	1	1	2	1.594	1.865
sqrt	2	2	4	3.189	3.730
sqrt	2	3	5	3.986	4.662
sqrt	3	4	7	5.580	6.527
int	1	1	1.613	2.903	2.043
int	2	2	3.226	5.806	4.086
int	2	3	4.033	7.257	5.107
int	3	4	5.646	10.160	7.150
log	1	1	1.135	4.019	2.096
log	2	2	2.270	8.037	4.192
log	2	3	2.837	10.046	5.240
log	3	4	3.972	14.065	7.336

XU4, which is also built around Exynos 5422, does not have this issue. It is important to note that we compiled all our benchmarks using the same `gcc` settings. We include this case of non-standard behaviour in our experiments to explore possible negative impacts on the performance modelling and optimization.

3) Core idle and active powers: The Odroid-XU3 board provides power readings per power domain, i.e. one combined reading per core type, from which it is possible to derive single core characteristic values w_0 and w.

Idle powers are determined by averaging over 1min of measurements while the platform is running only the operating system and the power logging software. The idle power values are $w_{0,\mathrm{A7}} = 0.1571\mathrm{W}$ and $w_{0,\mathrm{A15}} = 0.3552\mathrm{W}$, which are used across all benchmarks. The standard deviation during the idle power measurements is 1.1% of the mean value.

Effective powers $w_{\mathrm{A7}}, w_{\mathrm{A15}}$ are calculated from the measured active powers by subtracting idle power according to (30). The power ratios are then found as $\beta_{\mathrm{A7}} = 1$ and $\beta_{\mathrm{A15}} = w_{\mathrm{A15}}/w_{\mathrm{A7}}$; the values are presented in Table 2.

C. Amdahl's Workload Outcomes

A large number of experiments have been carried out covering all functions (sqrt, int, log) in various core configurations, and repeated for $p = 0.3$ and $p = 0.9$. This set of runs use a fixed workload of 60000 items with equal-share workload distribution between threads. Model predictions and experimental measurements for a set of selected homogeneous and heterogeneous multi-core configurations can be found in Figures 4 and 5. The measured speedup is calculated as the measured time for a single A7 core execution $t_{\mathrm{A7}}(1)$, shown in Table 2, over the benchmark's measured execution time $t(\overline{n})$:

$$S(\overline{n}) = \frac{t_{\mathrm{A7}}(1)}{t(\overline{n})}. \tag{46}$$

The observations validate the model (19) by showing that the differences between the model predictions and the experimental measurements are very small. The speedup error never exceeds 1%, and the power error never exceeds 4%, which is comparable to the standard deviation of the characterization measurements. A possible explanation for the low error values can be that our synthetic benchmark produces very stable $\overline{\alpha}$ and $\overline{\beta}$, and accurately emulates p. However, these small errors also prove that the model can be used with high confidence if it is possible to track these parameters. The model can also be confidently used in reverse to derive parallelization and performance properties of the system from the speedup measurements, as demonstrated in Section 7.

The counter intuitive result for 7-core (three A7 cores and four A15 cores) execution having lower power dissipation than four A15 cores and no A7 cores can be explained by the equal-share workload distribution. Because the parallel workload is equally split between these cores, the A15 cores finish early and wait for A7 cores. This idling reduces the average total power dissipation, however it implies that intelligent workload distribution can improve core utilization by scheduling more tasks to A15 cores than to A7 ones so that they finish at the same time. This is investigated in Section 6.5.

D. Gustafson's Workload Outcomes

Two sets of experiments have been carried out to validate heterogeneous Gustafson's models in both purely parallel and classical workload scaling modes described in Section 4.4. The initial workload I is set to 60000, and the scaled workload I' is defined by (20). Table 3 shows selected examples of calculating $g(\overline{n})$ for both scaling modes according to (25) and (28). These experiments also use equal-share workload distribution and s is fixed to A15. The integer $g(\overline{n})$ values in the case of sqrt are because $\min \overline{\alpha} = \alpha_s$ for this function.

The measured speedup is calculated as the ratio of performances according to (1), or as the time ratio multiplied by the workload size ratio:

$$S(\overline{n}) = \frac{t_{\mathrm{A7}}(1)}{t(\overline{n})} \cdot \frac{I'}{I}. \tag{47}$$

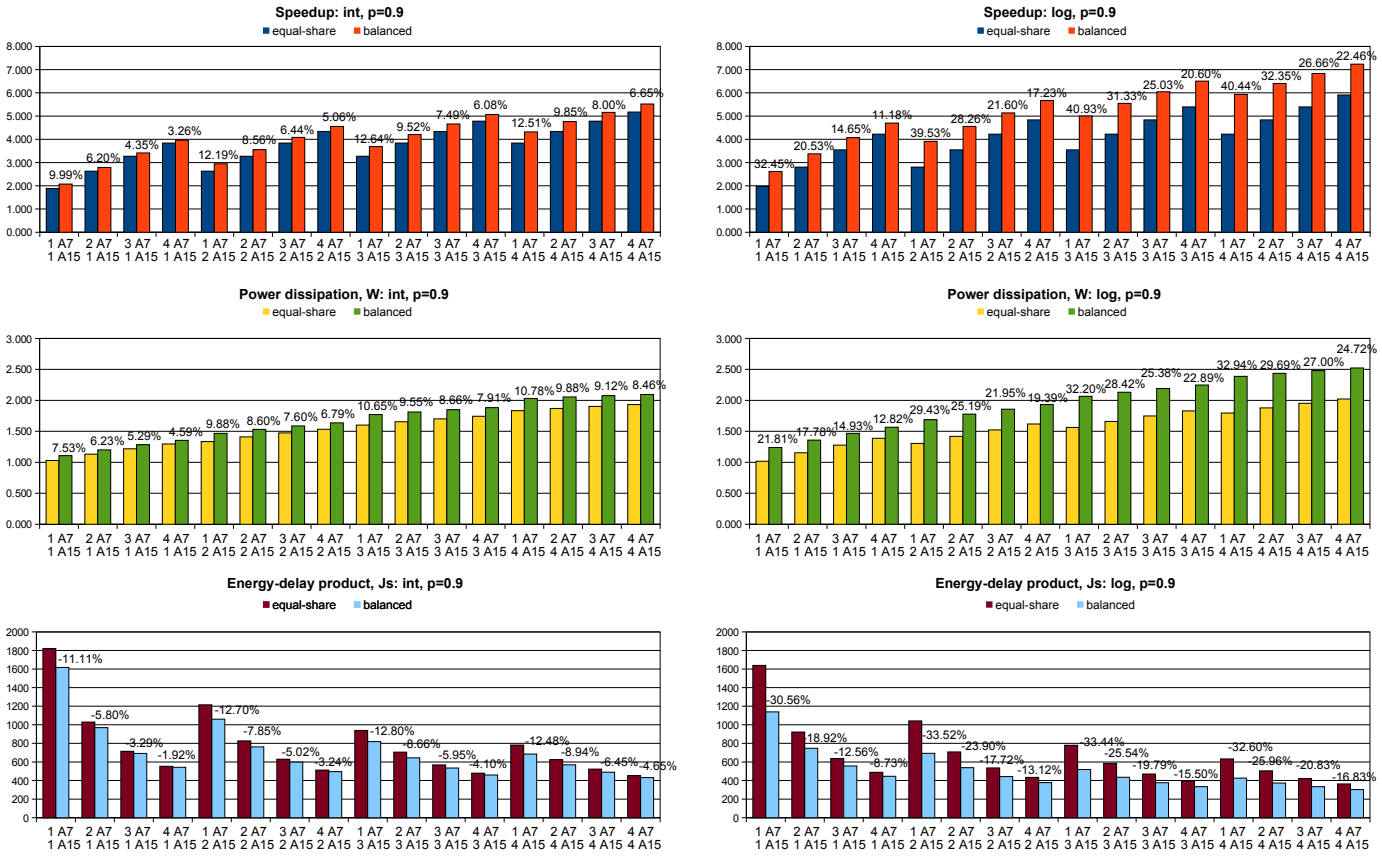

Fig. 7. Comparison of the speedup, power and energy between equal-share and balanced execution.

The observed errors are similar to the Amdahl's model with the speedup estimated within 1% error and the power dissipation estimated within 4% difference between the theory and the measurements.

Figure 6 compares the speedup between two workload scaling modes for $p = 0.3$. The purely parallel scaling has more effect for less parallelizable applications as it focuses on reducing the sequential part of the execution, hence the experiments with $p = 0.9$ show insignificant gain in the speedup and are not presented here. Even though the purely parallel scaling is harder to achieve in practice as it requires the knowledge of p, it provides a highly significant speedup gain, especially if the difference between the core performances is high, like in the case of log, which gives almost 50% better speedup. However, improper use may cause poor performance, as seen in the sqrt example using A15 for the sequential execution, which is slower than A7 for this function.

E. Balanced Execution

Previously described experiments use equal-share workload distribution, which is simpler to implement, but results in faster cores being idle while waiting for slower cores. The balanced distribution, defined in (17), gives the optimal speedup for a given workload. This section implements balanced distribution of a fixed workload and compares it to the equal-share distribution outcomes of Amdahl's law. The results are presented for $p = 0.9$, as it provides larger differences for this scenario.

In terms of model validation, the results are also very accurate, giving up to 4% error in power estimation and, in the most cases, within 1% error for the speedup. Slightly larger errors appear for the log benchmark in the cases of 7 or more cores, which show speedup errors of almost 4%. It is important to note that it is virtually impossible to achieve perfectly balanced workload distribution in real life, hence a slight mismatch between the benchmark execution and the theoretical model is to be expected.

Figure 7 explores the differences between the equal-share and optimal (balanced) cases of workload distribution in terms of performance and energy properties of the system, calculated according to Sections 4 and 5. The balanced distribution leads up to 41% increase in the speedup. The average power dissipation is also increased up to 33% as the cores are exercised with as little idling as possible. However, the power increase is not as big as the performance gain, hence the balancing of the workload is generally beneficial to the system, as can be seen from the calculated EDP metric.

On the other hand, the large differences in the speedup between these two cases pose a slight drawback to the practical use of the models. As discussed in the Section 4.1, these models form the corner cases for a load balancing algorithm, hence a larger range would reduce the predictive value of the method. However, in real life the experiments show that the load balancing indeed produces vastly different performance results as demonstrated in Section 7. Thus, the model predictions are correct, and narrowing the prediction range would

require detailed knowledge of the system load balancer.

VII. Real Application Workloads

This section is focused on experiments with realistic workloads based on the Parsec benchmark suite [24]. Parsec benchmarks are designed for parallel multi-threaded computation and include diverse workloads that are not exclusively focused on high performance computing. Each application is supplied with a set of pre-defined input data, ranging from small sizes (test) to large (simlarge) and very large (native) sizes. Each input is assumed to generate a fixed workload on a given system. To our knowledge, Parsec benchmarks do not implement workload scaling to Gustafson's or Sun-Ni's models, hence this section is focused on Amdahl's law only.

In our experiments we run a subset of Parsec benchmarks, namely ferret (CPU-heavy), fluidanimate (memory-heavy), and bodytrack (mixed), and use simlarge input. Core pinning of is done at the application level using the `taskset` command in Linux. The command takes a set of cores as an argument and ensures that every thread of the application is scheduled onto one of these cores. However, the threads are still allowed to move between the cores within this set due to the influence of the system load balancer [23]. This is different from the synthetic benchmark described in Section 6, which performed pinning of individual threads, one thread per core.

In this work, we do not study the actual algorithm of the load balancing or the internal structure of Parsec benchmarks, hence the workload distribution between the cores is considered a black box function: N_α is unknown. Section 4.1 addressed this issue by providing the range of values for N_α. The minimum value corresponds to equal-share workload distribution and gives the lower speedup limit $S_{\text{low}}(\overline{n})$; the maximum value is defined by the balanced workload and gives the higher speedup limit $S_{\text{high}}(\overline{n})$.

The goal of the following experiments is to calculate these limits and to find how the real measured speedup fits in the range. The relation provides a quality metric q for the load balancing algorithm, where $q = 1$ corresponds to the theoretically optimal load balancer, and $q = 0$ is equivalent to a nave approach (equal-share). Negative values may also be possible and show that the balancing algorithm is not working properly and creates an obstacle to the workload execution. The metric q is calculated as follows:

$$q = \frac{S(\overline{n}) - S_{\text{low}}(\overline{n})}{S_{\text{high}}(\overline{n}) - S_{\text{low}}(\overline{n})}. \tag{48}$$

The motivation for load balancing is to improve speedup by approaching the balanced workload behaviour. Hill-Marty [12] and related existing work [13], [14] covering core heterogeneity all assume that the workload is already balanced in their models, implying $q = 1$. This work makes no such assumption and studies real load balancer behaviours for different benchmarks, using novel models facilitating quantitative comparisons.

A. Model Characterization

The model characterization is obtained from the homogeneous configuration experiments, and then the models are used to predict system behaviour in heterogeneous configurations. Each benchmark is studied independently. Table 4 shows the obtained parameter values.

A7 is once again used as BCE, $\alpha_{\text{A7}} = 1$; α_{A15} values are derived from single core executions. Core frequencies of both A7 and A15 are set to 1.4GHz.

Parameter α_s is not known because it is not guaranteed that the sequential part of the workload will be executed on the fastest core, and it is also possible for the sequential execution to be re-scheduled to different core types, however α_s must stay within the range of $[\alpha_{\text{A7}}, \alpha_{\text{A15}}]$.

Parallelization factor p is determined from the measured speedup $S(n)$ for $n > 1$ using the equation:

$$p = \frac{n}{n-1}\left(\frac{1}{S(n)} - 1\right), \tag{49}$$

which is derived from the homogeneous Amdahl's law (3). For different values of n, the equation gives different p, however the differences are insignificant within the same type of core. On the other hand, the differences in p for different core types are substantial and cannot be ignored.

The lowest values of the model parameters p and α_s are used to calculate the lower limit of the heterogeneous speedup $S_{\text{low}}(\overline{n})$, and the highest values are used to calculate $S_{\text{high}}(\overline{n})$.

B. Quality of Load Balancer

Figure 8 presents the outcomes of the experiments for the selected benchmarks and heterogeneous core configurations. Time measurements have been collected from 10 runs in each configuration to avoid any random flukes, however the results were surprisingly consistent within 0.2% variability. This indicates that the system scheduler and load balancer behave deterministically in given conditions.

The graphs display the calculated speedup ranges $[S_{\text{low}}(\overline{n}), S_{\text{high}}(\overline{n})]$ and the measured speedup $S(\overline{n})$. The numbers represent the load balancer quality q, calculated from (48).

The first interesting observation is that the ferret benchmark is executed with incredible scheduling efficiency despite the system's heterogeneity. The average value of q is 0.81 and the maximum goes to 0.94. According to the benchmark's description, its data parallelism employs a pipeline, i.e. the application implements a producer-consumer paradigm. In this case, the workload distribution is managed by the application. Consequently, the cores are always given work items to execute and the longest possible idling time is less than the execution of one item.

The observed q values never exceed 1, which validates the hypothesis that (17) refers to the optimal workload distribution and can be used to predict the system's performance capacity. The lower bound of $q = 0$ is also mostly respected. This is not a hard limit, but a guideline that separates appropriate workload distributions. This boundary is significantly violated only in one case, as described below.

Bodytrack and fluidanimate show much less efficient workload distribution, compared to ferret, and their efficiency seems to decrease when the core configuration includes more little

TABLE IV
CHARACTERIZATION OF PARSEC BENCHMARK PARALLELIZABILITY FROM HOMOGENEOUS SYSTEM SETUP

	A7					A15					
app	$S(2)$	$S(3)$	$S(4)$	p_{A7}		$S(2)$	$S(3)$	$S(4)$	p_{A15}		α_{15}
bodytrack	1.8721	2.6499	3.3081	0.9320	$\pm$0.0020	1.8040	2.4606	3.0154	0.8910	$\pm$0.0006	1.9728
ferret	1.8772	2.6697	3.3786	0.9371	$\pm$0.0026	1.9192	2.7778	3.4648	0.9555	$\pm$0.0070	1.8795
fluidanimate	1.5726	–	2.2250	0.7311	$\pm$0.0029	1.4522	–	1.9422	0.6348	$\pm$0.0120	1.8164

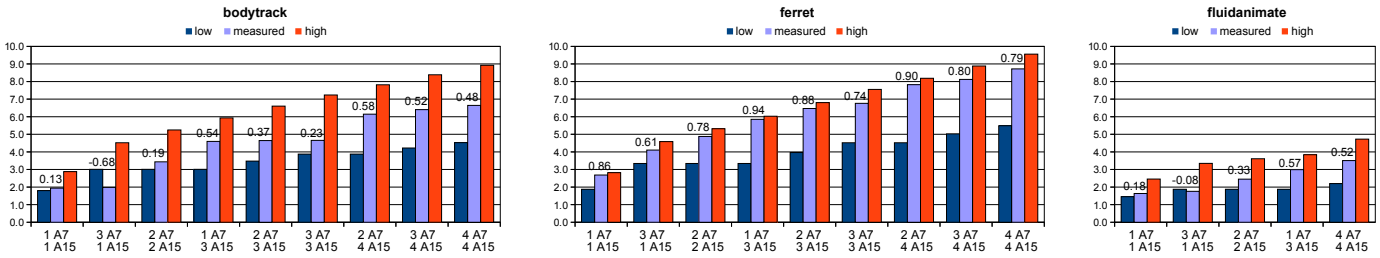

Fig. 8. Parsec speedup range results from heterogeneous system setup determining q – the quality of the system load balancer

than big cores. This effect is exceptionally impactful in the case of three A7 cores and one A15 core executing four threads of the bodytrack application. The value of q for this configuration lies far in the negative range and can serve as an evidence of load balancer malfunction. Indeed, the speedup of this four-thread execution is only slightly higher than two-threaded runs on one A7 and one A15. The execution time is close to a single thread executed on one A15 core, showing almost zero benefit from bringing in three more cores, and the result is consistent across multiple runs of the experiment. This issue requires a substantial investigation and lies beyond the scope of this paper, however it demonstrates how the presented method may help analyse the system behaviour and detect problems in the scheduler and load balancer.

VIII. CONCLUSIONS

The models presented in the paper enhance our understanding of scalability in heterogeneous many-core systems and will be useful for platform designers and electronic engineers, as well as for system level software developers.

This paper extends three classical speedup models – Amdahl's law, Gustafson's model and Sun Ni's model – to the range of heterogeneous system configurations that can be described as a normal form heterogeneity. This type of heterogeneity is defined and analysed with respect to the model assumptions. The analysis includes an in-depth discussion of possible model limitations and shows that the proposed models are not reducing applicability in comparison to the original models. The provided discussion can serve as a foundation for multiple research directions in the future. Important aspects, such as workload distribution between heterogeneous cores and various modes of workload scaling, are included in the model derivation. In addition to performance, this paper addresses the issue of power and energy modelling by calculating power dissipation, energy consumption, and energy delay product for the respective heterogeneous speedup models.

The practical part of this work includes experiments on the Odroid-XU3 board pertaining to model validation and real-life application. The models have been validated against a synthetic benchmark in a controlled environment. The experiments confirm the accuracy of the models and show that the models provide deeper insights and clearly demonstrate the effects of various system parameters on performance and energy scaling in different heterogeneous configurations.

The modelling method enables the study of the quality of load balancing, used for improving speedup. A quantitative metric for load balancing quality is proposed and a series of experiments involving Parsec benchmarks are conducted. The modelling method provides quantitative guidelines of load balancing quality against which experimental results can be compared. The Linux load balancer is shown to not always provide high quality results. In certain situations it may even produce worse results than the nave equal-share approach. The study also showed that application-specific load balancing using pipelines can produce results of much higher quality, approaching the theoretical optimum obtained from the models.

ACKNOWLEDGEMENT

M. A. N. Al-hayanni thanks the Iraqi Government for PhD studentship funding. The authors thank Ali M. Aalsaud for useful discussions.

REFERENCES

[1] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[2] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[3] X.-H. Sun and L. M. Ni, "Another view on parallel speedup," in *Supercomputing'90., Proceedings of.* IEEE, 1990, pp. 324–333.

[4] ——, "Scalable problems and memory-bounded speedup," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 27–37, 1993.

[5] S. Borkar, "Thousand core chips: A technology perspective," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 746–749. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278667

[6] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[7] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *Annals of the History of Computing, IEEE*, vol. 33, no. 3, pp. 46–54, 2011.

[8] F. J. Pollack, "New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)," in *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 1999, p. 2.
[9] P. Greenhalgh, *big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7 – Improving Energy Efficiency in High-Performance Mobile Platforms*, ARM, 2011, white Paper.
[10] "Juno ARM development platform SoC technical overview," ARM, Tech. Rep., 2014. [Online]. Available: http://www.arm.com
[11] R. H. Dennard, V. Rideout, E. Bassous, and A. Leblanc, "Design of ion-implanted mosfet's with very small physical dimensions," *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 5, pp. 256–268, 1974.
[12] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, no. 7, pp. 33–38, 2008.
[13] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, pp. 183–188, 2010.
[14] N. Ye, Z. Hao, and X. Xie, "The speedup model for manycore processor," in *Information Science and Cloud Computing Companion (ISCC-C), 2013 International Conference on*. IEEE, 2013, pp. 469–474.
[15] D. H. Woo and H.-H. S. Lee, "Extending amdahl's law for energy-efficient computing in the many-core era," *Computer*, no. 12, pp. 24–31, 2008.
[16] M. A. N. Al-Hayanni, A. Rafiev, R. Shafik, and F. Xia, "Power and energy normalized speedup models for heterogeneous many core computing," in *16th International Conference on Application of Concurrency to System Design (ACSD)*, June 2016, pp. 84–93.
[17] "Odroid XU3," http://www.hardkernel.com/main/products.
[18] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.
[19] J. Baeten, C. A. Middelburg, and E. T. Netherlands, "Process algebra with timing: Real time and discrete time," in *Handbook of Process Algebra*. Elsevier, 2000, pp. 627–684.
[20] A. B. Downey, "A model for speedup of parallel programs," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-97-933, Jan 1997. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1997/5394.html
[21] K. Georgiou, S. Kerrison, Z. Chamski, and K. Eder, "Energy transparency for deeply embedded programs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 1, pp. 1–26, 4 2017.
[22] X. Li and M. Malek, "Analysis of speedup and communication/computation ratio in multiprocessor systems," in *Proceedings. Real-Time Systems Symposium*, Dec 1988, pp. 282–288.
[23] J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, and A. Fedorova, "The linux scheduler: A decade of wasted cores," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: ACM, 2016, pp. 1:1–1:16. [Online]. Available: http://doi.acm.org/10.1145/2901318.2901326
[24] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

Ashur Rafiev has received his PhD in 2011 in the School of Electrical, Electronic and Computer Engineering, Newcaste University. At the moment, he works in the School of Computing Science, Newcastle University, as a Research Associate. His research interest is focused on power modelling and hardware-software co-simulation of many-core systems.

Mohammed Al-hayanni (Student Member, IEEE and IET) is an experienced electronics, computer and software engineer. He is currently studying for his PhD with the School of Electrical and Electronic Engineering, Newcastle University. His research interests include developing practically validated robust performance adaptation models for energy-efficient many-core computing systems.

Fei Xia is a Senior Research Associate with the School of Electrical and Electronic Engineering, Newcastle University. His research interests are in asynchronous and concurrent systems with an emphasis on power and energy. He holds a PhD from King's College, London, an MSc from the University of Alberta, Edmonton, and a BEng from Tsinghua University, Beijing.

Rishad Shafik (MIEE06-to-date) is a Lecturer in Electronic Systems at Newcastle University. His research interests include design of intelligent and energy-efficient embedded systems. He holds a PhD and an MSc from Southampton Uni., and a BEng from IUT, Bangladesh. He has authored 80+ research articles published by IEEE/ACM, and is the co-editor of "Energy-efficient Fault-Tolerant Systems". He is chairing DFT'17 (http://www.dfts.org), to be held in Cambridge, UK.

Alexander Romanovsky is a Professor at Newcastle University, UK and the leader of the Secure and Resilient Systems group at the School of Computing Science there. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system verification for safety, system structuring and verification of fault tolerance. He is a member of the editorial boards of Computer Journal, IEEE Transactions on Reliability, Journal of System Architecture and International Journal of Critical Computer-Based Systems.

Alex Yakovlev is a professor in the School of Electrical and Electronic Engineering, Newcastle University. His research interests include asynchronous circuits and systems, concurrency models, energy-modulated computing. Yakovlev received a DSc in Engineering at Newcastle University. He is Senior Member of IEEE and Fellow of IET. In 2011-2013 he was a Dream Fellow of the UK Engineering and Physical Sciences Research Council (EPSRC).