Optimum Scalability Point for Parallelisable Real-Time Components

Tommaso Cucinotta
Real-Time Systems Laboratory
Scuola Superiore Sant'Anna
cucinotta@sssup.it

***Abstract*—Distributing the workload of computationally intensive software components across a set of homogeneous computing resources (nodes, hosts, processors, cores), for the purpose of allowing them to meet precise timing (response-time) constraints, is often a pain due to the difficulties in understanding how the software will actually scale. Often, such a problem is faced by recurring to a trial-and-error process. In this paper, a methodology is introduced to tackle the problem of finding the optimum number of processors for deploying parallelisable real-time software components. Basic building blocks of the methodology are: a generic performance model for the response-time of a parallel software component; a concrete procedure for tuning optimally the parameters of the model; the application of optimisation techniques that allow to compute what is the minimum allocation needed to meet precise deadline constraints, as well as the one that minimises the response-time; the consideration of multiple real-time (independent) components to be deployed on the same multi/many-core hardware.**

Acknowledgements

I would like to thank: Lutz Schubert from the HLRS of Stuttgart for having helped me in identifying appropriate related works for this paper; the European Community's Seventh Framework Programme FP7 which funded this research under grant agreement n.248465 S(o)OS – Service-oriented Operating Systems; and the anonymous reviewers who contributed to improve the final revision of this paper.

I. Introduction and Related Work

A number of real-time applications have nowadays so high computing requirements and at the same time so strict timing constraints that they need to be deployed over multi-core, and sometimes high-performance computing platforms. Optimum (high-level) control of robots and complex on-the-fly transformations of high-definition media, such as required in systems for video surveillance or video ingesting and editing in the film post-production industry, constitute possible scenarios of this kind. However, besides the complex issues behind parallelising the algorithms and software, system designers are also faced with the problem of how to optimally tune the hardware platform so as to deploy this kind of applications in such a way that their temporal constraints are met, or also how to optimally assign resources for a multitude of applications of this kind that will run over the same many-core, massively parallel and distributed system. In order to properly face with these issues, it is necessary to model behaviour of a parallelisable application as a function of the number of computing elements over which it may be deployed.

Attempts to formalise the performance of parallel software components highlighting their scalability properties in the number of processors date back to year 1967, with the well-known Amdahl law [2], cited many times as a *pessimistic* result in the domain of parallel computing due to the upper bound to the achievable speed-up no matter how many processors are available. Later, Gustafson tried a reinterpretation [17] of the law after experimenting with 1024 processors introducing something called *scaled speed-up*, claimed to justify the results apparently opposing to the Amdahl's argument. In the end, it turns out that the two formulations were identical, as shown e.g., by Shi [25], who also stressed on the role played by an algorithm own complexity in the data size, when evaluating parallel speed-ups. Indeed, Shi underlined that these speed-up models are valid only if the algorithm is *structure-persistent*, i.e., its parallel version has the same number of computing steps as the sequential one (for example, a quadratic sort would break this rule and turn out to possess a super linear speed-up). Back on the claimed pessimism of the Amdahl's law, it turns out that actually there was even too much *optimism* therein, as the fundamental overheads due to distribution, synchronisation and communication among the parallel parts of the code had been completely neglected. Therefore, many authors introduced an additional term in the execution time formula, having a linear (or sometimes quadratic) dependency on the number of processors [8], [29]. The immediate consequence of this, as nicely argued by Brown [8], [27], is that increasing a parallel software performance requires a careful evaluation not only of the number of processors that we may add to the platform, but also of the possibilities in enhancing communications (e.g., switching to higher performance network adapters), so as to reduce the impact of one of the most important factors limiting scalability. Another interesting variation proposed by Sun and Ni [27] is the one to not limit the speed-up model to consider merely completely sequential or completely parallelisable code sections. Instead, it is possible to consider that various segments in the code may scale well only up to a segment-specific maximum number of processors. Also, Turek et al. [28] investigated on how to minimise the average response-time of a set of parallelisable tasks.

An interesting point made by various authors, e.g., [27], concerns the distinction between the classical speed-up factor,

measured at equal work to be done between the parallel and sequential version of a software and the time-bounded speed-up measured at equal available processing time (a memory-bounded speed-up may be found as well). The latter barely recalls the problems investigated in the domain of multi core real-time systems, where the interest is in how to make a piece of software respect a given deadline constraint.

Li and Malek [22] actually tackled the problem of minimum and optimum processors assignment to a parallelisable real-time application under linear communication overheads, accounting for both a sequential and a parallel model of communications. Such a model is among the ones considered in the present paper, so some of their original results are recalled later (useful for the difference in notation). However, more models for the distribution overheads are considered in this paper, and the presented methodology may generically deal with additional models as well.

Other works tried to model the behaviour of parallel computing systems to the purpose of studying the performance of parallel algorithms implemented on top of it. One of the classical models is the Parallel Random-Access Machine by Fortune and Willie [15] (PRAM), which left space for various extensions, for example to cope with communication and contention overheads in distributed or multi-processor systems. For example, the LogP approach by Culler et al. [13] properly includes in the machine model the communication latency and bandwidth. Also, there exists recent work investigating on the trade-offs between achievable speed-up, computing power of heterogeneous cores in multi-core chips, and design costs for realising them, such as the one by Hill and Marty [18].

There is a wide literature in the real-time community on scheduling tasks on multi-processor and multi-core platforms, e.g., [3], [6], [4], [5], [7], [11], [21], [26], [23], [20], [24] just to mention a few. These works focused mostly on algorithms for temporal scheduling and schedulability analysis techniques for distributed real-time task models with mostly a static structure (i.e., where the number of tasks to deploy and their characteristic is known and fixed). Some of these works addressed specifically the problem of allocation of real-time applications in a distributed environment [14], [19], [23], but not with a variable parallelism degree for the applications. Also, many authors focused on the optimum and adaptive allocation options for real-time tasks [9], [12], [1], [10], [16], but usually with a focus on the best scheduling parameters of a single real-time task, rather than the number of processing units over which it may be parallelised. Adaptive techniques for on-line setting of the parallelism degree have been employed also in the domain of parallel computing [29], and the results from this work might integrate in an adaptive policy as well, if the execution times of the model are continuously estimated/refined on-line instead of being profiled ahead of time. For space reasons, further related works cannot be mentioned.

II. Approach

A. Assumptions

For the sake of simplicity, in the following it is assumed that a number of homogeneous computing elements $\mathcal{H} = \{1, \ldots, N_H\}$ are available for deploying a set of distributed real-time applications. Each computing element might be a host, a processor in a multi-processor machine, or a core in a multi-core machine. Without loss of generality, these terms will be used interchangeably from here on. Clearly, these different options correspond to different programming paradigms exploitable by developers to code the interactions (synchronisation and communication) among the concurrently executing parts of an algorithm. However, in this paper the impact of these interactions on the performance of the application, and specifically on its response-time, is modelled in a general and abstract way which may be equally representative of all of the mentioned cases. Also, the discussion that follows fits optimally when the considered set of computing elements not only is homogeneous, but also the latencies across the elements are homogeneous. For example, if the computing units under consideration reside all in separate hosts belonging to the same subnet, or if they are all different cores of the same multi-core machine. In case one wanted to consider both of these cases, a rough approximation is the one in which the communication penalties due to the faster links are approximated (roughly) as equal to the ones due to the slowest links. In the future, this work might be extended to consider more thoroughly this problem.

B. Parallel Performance Model

Consider a set of parallelisable real-time software components (or applications) $\mathcal{A} = \{1, \ldots, N_A\}$. Each application $a \in \mathcal{A}$ is characterised by the following response-time model, when deployed over $x \in \mathbb{N}$ dedicated identical processors:

$$R_a(x) = \frac{P_a}{x} + S_a + O_a(x), \tag{1}$$

where $P_a \in \mathbb{R}^+$ is the ideally perfectly parallelisable part of its computation time (scaling with the number of processing units), $S_a \in \mathbb{R}^+$ is the sequential, non-parallelisable part of it and $O_a(x)$ is the additional penalty in the execution time to be paid for overheads due to the additional communications among the distributed parts of the component, including distribution and results joining overheads, assumed to be a non-decreasing function of the number of units over which the workload is distributed, beyond the first one (i.e., $O_a(1) = 0$). Note that $\frac{S_a}{S_a+P_a}$ is the fraction of the original sequential computation that is assumed to not be parallelisable. From here on, whenever the discussion refers to a single application $a \in \mathcal{A}$, its index will be omitted for notational convenience.

Two basic communication overhead models are considered in this paper: *linear* and *logarithmic* overheads.

1) Linear Communications Overheads: For the well-known linear model we have $O(x) = K(x-1)$, with K constant, thus:

$$R(x) = \frac{P}{x} + S + K(x-1). \tag{2}$$

Figure 1(a) graphically depicts various $R(x)$ curves obtained at varying values of the communication overhead K, and Figure 1(b) reports the corresponding *relative speed-up*, defined as the ratio between the response-time with 1 processor and the

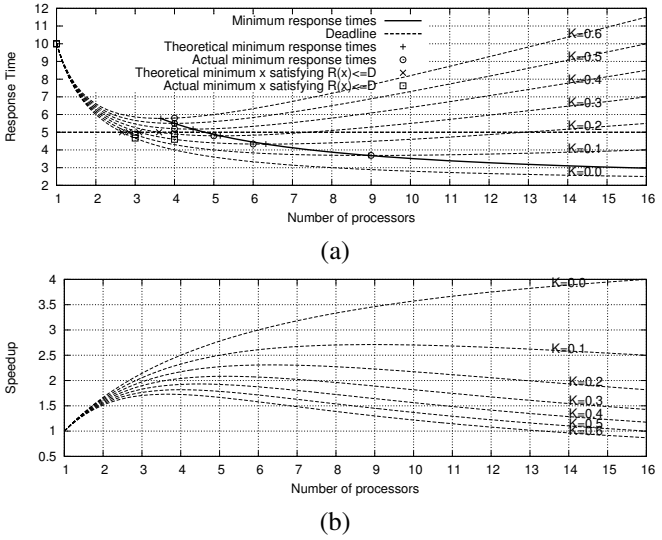

Figure 1. Response times (a) and speed-up (b) corresponding to the model in Equation (1), as a function of the parallelism degree x (along the X axis), with $P = 8$ and $S = 2$ (i.e., a non-parallelisable factor of 20%) and at various values of K (corresponding to the different curves). The top figure reports also the curve identifying the minimum response-times and corresponding parallelism degree as from Equation (8) and a sample deadline of $D = 4$.

one with x processors. With $K = 0$ the well-known Amdahl's Law [2] is obtained: as $x \to \infty$, the response time keeps decreasing towards S (and the speed-up increases towards $1 + P_a/S_a$). However, for $K > 0$, there exists a maximum number of processors (corresponding to the points marked in the picture with plus signs) beyond which the response time starts increasing again (and the speed-up starts decreasing), due to the communication overheads in Equation (1) that starts dominating the other terms, bringing $R(x) \to \infty$ as $x \to \infty$.

Also, each application is assumed to be associated with a timing constraint, i.e., its maximum allowed response time: $R(x) \leq D$. Clearly, the focus of the paper is on the need for deploying a software component on a number of processors strictly greater than 1, in order to be able to meet such constraint. For example, in Figure 1(a) it can be clearly seen that, in order to meet the $R(x) \leq D = 5$ deadline constraint, at least 3 processors would be needed if $K \leq 0.1$, however at least 4 would be needed if $K \in [0.2, 0.3]$, and finally respecting the deadline becomes impossible for $K \geq 0.4$. Additionally, the focus of the investigation is on the impact of the communication and distribution overhead K over the software performance, thus K is assumed to be strictly positive. Therefore, in the discussion that follows, these conditions will be implicitly satisfied, unless otherwise stated (this allows us to skip a few border-line cases of little interest):

$$S \leq D < R(1) \equiv P + S \wedge K > 0. \tag{3}$$

a) Parallelism Degree: Now the problem of deciding a proper parallelism degree for the application is formalised and solved. Specifically, two sub-problems are considered:

- find the *minimum parallelism degree* by which a given response-time constraint D for the application is satisfied;
- find the *optimum parallelism degree* by which the obtained response-time is minimised.

In what follows, the two problems are tackled separately, then a possible integration of both of them in a general methodology for deploying a set of scalable real-time applications is addressed in Section II-D.

b) Minimum Parallelism Degree: Given the notation and assumptions introduced above, we can now state the following:

Proposition 1. *A necessary condition for the existence of a number of processors $x \in \mathbb{N}$ allowing the application to meet a given response-time constraint D is:*

$$D \geq S - K + 2\sqrt{KP} \tag{4}$$

Also, if such a number exists, then its minimum value is:

$$x^{(min)} = \left\lceil \frac{D - (S-K) - \sqrt{[D-(S-K)]^2 - 4KP}}{2K} \right\rceil \tag{5}$$

Finally, a sufficient condition for $x^{(min)}$ to exist (thus allowing the application to respect its timing constraint) is:

$$D \geq S - K + 2\sqrt{KP}\sqrt{1 + \frac{K}{4P}} \tag{6}$$

Proof: This result can easily be obtained by studying the inequality $R(x) \leq D$:

$$\frac{P}{x} + S + K(x-1) \leq D \tag{7}$$

that can easily be stated in terms of a second order inequality: $Kx^2 - [D - (S-K)]\,x + P \leq 0$. In this case, it is easy to see that, if the left side of this inequality has no real zeros (i.e., Equation (4) if violated), then no solution can exist. Now, it is easy to verify that $\tilde{x} = \frac{D-(S-K)-\sqrt{[D-(S-K)]^2-4KP}}{2K}$ is the minimum real zero of Inequality (7). However, we are searching for a parallelism degree that has an integer value, thus either we are lucky and $\tilde{x} \in \mathbb{N}$ thus $x^{(min)} = \lceil \tilde{x} \rceil = \tilde{x}$, or (more likely) $\tilde{x} \notin \mathbb{N}$. Then, we must round it up, but there are two cases: $\lceil \tilde{x} \rceil$ is lower than or equal to the other zero, thus such value still respects the deadline constraint, or $\lceil \tilde{x} \rceil$ is higher than the other zero and the problem has no solutions. Imposing that the distance between the two real solutions be greater than or equal to 1, i.e., Equation (6), ensures that at least one integer exists in the interval, and $x^{(min)} = \lceil \tilde{x} \rceil$ is the minimum among such integers. ■

c) Optimum Performance Parallelism Degree: While the minimum parallelism degree $x^{(min)}$ as stated in Equation (5) is sufficient for satisfying the response-time constraint D, if additional resources are available, then it may be worth to investigate on up to which point it is convenient to pull additional resources for deploying the real-time application. Indeed, one of the consequences of the response-time model under consideration as stated in Equation (1) is that, for $x \to \infty$, the response-time goes to infinity as well (under the assumption that K is strictly positive).

The problem of finding the parallelism degree achieving the maximum performance (i.e., minimum response-time) can be conveniently investigated by minimising the obtained response-time in Equation (1) along the x variable.

Proposition 2. *The minimum response-time achievable by parallelising the application under the model in Equation (1), which exists only if $K > 0$, is bounded by:*

$$R^{(min)} \geq S - K + 2\sqrt{KP}$$

and it is achieved by the following number of processors:

$$x^{(opt)} = \begin{cases} \left\lfloor \sqrt{\frac{P}{K}} \right\rfloor & if\ R\left(\left\lfloor \sqrt{\frac{P}{K}} \right\rfloor\right) \leq R\left(\left\lceil \sqrt{\frac{P}{K}} \right\rceil\right) \\ \left\lceil \sqrt{\frac{P}{K}} \right\rceil & otherwise. \end{cases} \quad (8)$$

Proof: It is easy to verify the statement imposing that the derivative in x of Equation (1) be null, that leads to $\tilde{x} = \sqrt{\frac{P}{K}}$. It is also straightforward to verify that such point corresponds to a minimum of the function. The $\tilde{x}$ values and their corresponding response-times are highlighted in Figure 1(a) by means of plus signs. However, we are faced with the problem of choosing a proper *integer* $x^{(opt)}$ value as close as possible to $\tilde{x}$, minimising the response-time. The only way to discriminate among the two integers $\lfloor \tilde{x} \rfloor$ and $\lceil \tilde{x} \rceil$ is to compare the result of Equation (1) in correspondence of the two values, with a preference for the lowest x in case of parity. The obtained $x^{(opt)}$ are shown in Figure 1(a) by means of small circles. ■

Note that the $\tilde{x}$ formulation is equivalent to the one found by Li and Malek as appearing in Equation (9) of [22]. However, the surrounding arguments constitute still an interesting view on the problem, especially when considering the achievable optimum point between the minimum allocation of Equation (5) and the optimum one of Equation (8), in view of a multitude of applications to host on the available resources, as shown in Section II-D.

2) Logarithmic Communications Model: In certain parallel algorithms it may be possible to properly design the communications among the processors for distribution, synchronisation and results collection in such a way that they impact only logarithmically on the overall execution time. This is possible, for example, if the physical topology is arranged hierarchically and the workload distribution pattern exploits the topology so as to maximise the parallel communications during the distribution, aggregation and execution phases. For example, a balanced tree of computations may be properly set-up, so that communications for status updates among the nodes may be done concurrently, and in such a way that parent nodes may aggregate the received information from the children, resulting in an overall communication overhead that grows with the tree depth, i.e., logarithmically with the number of involved processors, i.e.:

$$R(x) = \frac{P}{x} + S + H \log x \quad (9)$$

Figure 2 depicts graphically this model, along with the significant points, similarly to what done for the linear overhead case. Now, we can state the following:

Proposition 3. *The minimum response-time achievable under the overhead model in Equation (9) is bounded by:*

$$R^{(opt)} \geq H + S + H \log \frac{P}{H}.$$

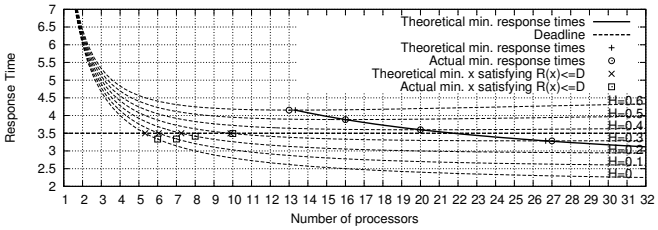

Figure 2. Response times corresponding to the model in Equation (9), as a function of the parallelism degree x (along the X axis), with $P = 8$, $S = 2$, $D = 3.5$, and at varying values for H (corresponding to the various curves).

The allocation that achieves the minimum response-time is:

$$x^{(opt)} = \begin{cases} \left\lfloor \frac{P}{H} \right\rfloor & if\ R\left(\left\lfloor \frac{P}{H} \right\rfloor\right) \leq R\left(\left\lceil \frac{P}{H} \right\rceil\right) \\ \left\lceil \frac{P}{H} \right\rceil & otherwise. \end{cases} \quad (10)$$

Proof: This is easily obtained by computing the point of null derivative of Equation (9). ■

Proposition 4. *The minimum allocation $x^{(min)}$ that respects the deadline, if it exists, can be found by the following recursive equation:*

$$\begin{cases} x_0 = \frac{P}{2H} \\ x_{i+1} = \frac{P}{D - S - H \log x_i} \quad i = 0, 1, \ldots \end{cases} \quad (11)$$

then setting $x^{(min)} = \lceil \lim_{i \to \infty} x_i \rceil$, where from a practical standpoint the recursion can be stopped as soon as x_i is detected to be converging, i.e.: $|x_{i+1} - x_i|$ is lower than a desired precision threshold.

Proof: Imposing that $R(x) = D$ results unfortunately in a transcendental equation whose solution amounts to finding the fixed point of $x = \frac{P}{D - S - H \log x}$. For example, this can be approximated with the mentioned method. ■

Note that the optimum and minimum allocation points highlighted in Figure 2 have been obtained with the above formula and recursive algorithm (stopping after 4 steps), respectively. However, the exact conditions under which the recursive Equation (11) converges deserve to be exactly identified.

C. Model Fitting

It is interesting to investigate on how we may optimally tune the model parameters P, S, K and H as needed in the models introduced above. The models in Equations (2) and (9) fall in the general class of linear combination of arbitrary functions $R(x) = \sum_{i=1}^{n} a_i f_i(x)$, thus it is sufficient to perform various measurements with different number of processors then minimise the average square error by solving a system of linear equations. The procedure is widely known, however it is recalled in what follows for the sake of completeness.

Assume to have available k measurements of the response-time r_j obtained with x_j processors ($j = 1, \ldots, k$). The parameters $\{a_i\}_{i=1}^{n}$ can be found as the ones minimising the average square *relative* error between the response times obtained from the model and the actually measured values: $E = \frac{1}{k} \sum_{j=1}^{k} \left(\frac{\sum_{h=1}^{n} a_h f_h(x_j) - r_j}{r_j} \right)^2$. Considering the relative

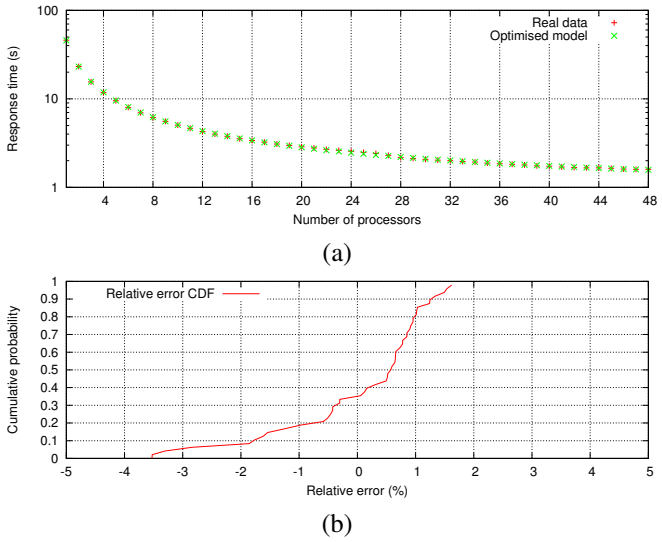

Figure 3. Results obtained after optimising the model in Equation (2) for `pbzip2` while compressing the Linux kernel sources. The model response times are compared with the real data in (a), whilst the cumulative distribution over the relative error after the optimisation is shown in (b).

error allows for avoiding polarisation of the optimisation towards the fewer cores measurements, which exhibit higher response times. Searching for the point of null gradient (with respect to the $\{a_i\}$ variables) leads to $\frac{\partial E}{\partial a_i} = 0 \iff \sum_{h=1}^{n}\left[\sum_{j=1}^{k}\frac{f_h(x_j)f_i(x_j)}{r_j}\right]a_h = \sum_{j=1}^{k} f_i(x_j)$, which is a linear system with n unknowns and n equations that can easily be solved. Note that at least $k \geq n$ independent measures (i.e., corresponding to different processors numbers) are needed in order for the system to have a unique solution. If more readings are available for the same parallelism degree x, all of them can be included in the above system coefficients, so as to enhance the precision of the model.

For example, running the just described procedure for modelling the execution times of `pbzip2`[1] while compressing the Linux kernel sources (with a compression level of 9) over a Dell PowerEdge R815 server equipped with 4 AMD Opteron 6100 processors (for a total of 48 cores) leads to the results shown in Figure 3. Figure 3(a) reports the originally gathered data versus the ones interpolated through the model. Figure 3(b) shows the cumulative distribution of the relative error over the collected samples, highlighting that it resides for 90% of the cases between +/-2%.

D. Deploying Multiple Applications

In the above sections, a single parallelisable real-time application was modelled and its optimum deployment options analysed in isolation. When dealing with multiple applications of this kind (with heterogeneous parameters) to be deployed over a set of available physical resources, a means is needed to perform an optimum deployment of the applications.

To this purpose, consider a set of N_A applications to be deployed over N_H computing units. As mentioned above, looking at the resource allocation x_a for each individual application, the minimum value that is needed in order to meet the response-time constraint is dictated by Equation (5), and the maximum value beyond which the performance starts to degrade is stated in Equation (8). Furthermore, the total availability of resources N_A needs to be considered. Therefore, the following set of constraints is obtained:

$$\begin{cases} x_a & \geq x_a^{(min)} \\ x_a & \leq x_a^{(opt)} \\ \sum_{a=1}^{N_A} x_a & \leq N_H. \end{cases} \tag{12}$$

A solution to this problem may easily be found by construction. First, the problem has no admissible solutions if there are not enough resources for the minimum allocations, thus $N_H \geq \sum_{a=1}^{N_A} x_a^{(min)}$ must hold true. Then, the residual available resources $N_H - \sum_{a\in\mathcal{A}} x_a^{(min)}$ may be distributed in a number of ways, for example with a linear redistribution giving more resources to components that are further away from their optimal response-time point, such as:

$$\hat{x}_a = x_a^{(min)} + \left(N_H - \sum_{a=1}^{N_A} x_a^{(min)}\right)\frac{x_a^{(opt)} - x_a^{(min)}}{\sum_{b=1}^{N_A}\left(x_b^{(opt)} - x_b^{(min)}\right)}. \tag{13}$$

However, the just computed values $\{\hat{x}_a\}_{a\in\mathcal{A}}$ still suffer of two potential problems:

1) they may represent an infeasible solution, i.e., they represent an excessive allocation that goes beyond the optimum point for the components $\hat{x}_a > x_a^{(opt)}\ \forall a$ (if this happens, then it happens for all the components); in such a case, it is sufficient to set $x_a = x_a^{(opt)}\ \forall a$.
2) they may be non-integers. Therefore, some of them will have to be rounded up, others rounded down, so as to achieve the final allocation. Note that, due to the constraints posed on the problem, rounding down all the values leads to an admissible solution.

In the latter case, an algorithm achieving the final allocations $\{\hat{x}_a\}$ may be, for example, the following:

1) set $A = \mathcal{A}$ and N_H equal to the number of processors;
2) set $H \leftarrow N_H - \sum_{a\in A} x_a^{(min)}$ and $\forall a \in A$, $\hat{x}_a \leftarrow x_a^{(min)} + H\frac{x_a^{(opt)} - x_a^{(min)}}{\sum_{b\in A}\left(x_b^{(opt)} - x_b^{(min)}\right)}$;
3) choose $a \in A$ for rounding down its allocation: $\hat{x}_a \leftarrow \lfloor \hat{x}_a \rfloor$;
4) update N_H and A according to: $N_H \leftarrow N_H - \hat{x}_a$ and $A \leftarrow A \setminus \{a\}$;
5) repeat from step 2 until $A = \emptyset$.

At step 3, any criterion may be chosen to select the next component whose allocation will be rounded down. For example, the first element (or a random element) in the A set may be chosen, or the component that looses the least in terms of response-time may be chosen, i.e., a s.t. $|R_a(\hat{x}_a) - R_a(\lfloor \hat{x}_a \rfloor)|$ is minimum. Note also that, at step 2, the re-computation of the residual available hosts as well as the $\hat{x}_a$ values after having finalised the allocation for each component allows for accounting for all the unallocated residuals due to rounding from the previous steps, so that the final allocation will result in $\sum_{a\in\mathcal{A}} x_a \geq N_H - 1$.

Considering a generic example with 2 components, the allocation problem can be graphically depicted, as shown

[1]More information is available at: http://compression.ca/pbzip2/.

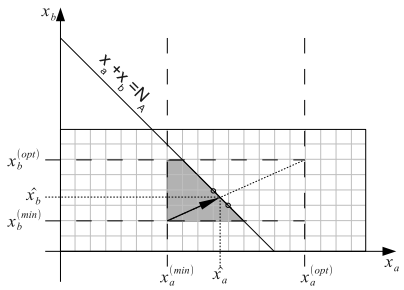

Figure 4. Visualisation of the allocation problem for two components.

in Figure 4. The allocation variables x_a and x_b vary along the X and Y axes of the figure, respectively. The graph shows the $45°$ line going from up-left towards bottom-right delimiting the pairs achieving saturation of the available hosts $x_a + x_b = N_H$. Also, the two schedulability regions for the two applications, $x_a \in [x_a^{(min)}, x_a^{(opt)}]$ and $x_b \in [x_b^{(min)}, x_b^{(opt)}]$, are visible. The intersection of these three areas is highlighted in grey, showing the set of all the admissible solutions to the problem. As highlighted by the black arrow, choosing a solution close to the point identified in Equation (13) amounts to move linearly from the $(x_a, x_b) = (x_a^{(min)}, x_b^{(min)})$ point towards the $(x_a, x_b) = (x_a^{(opt)}, x_a^{(opt)})$ point, till the intersection with the saturation constraints. Only pairs with integer coordinates, shown as points of the visible grid, are the admissible solutions. Therefore, it is not possible to use the $(x_a, x_b) = (\hat{x}_a, \hat{x}_b)$ solution, but we need to choose one of the grid points on the saturation constraint line closest to it, marked through small circles on the graph).

III. Conclusions and Future Work

In this paper, a comprehensive methodology for optimum tuning of the number of processors to allocate to a set of parallelisable real-time components has been presented. This may constitute a valuable receipt for practitioners approaching this kind of problems. Apart from recalling (and slightly refining) a few results already known from the past, a logarithmic overhead model was introduced that is suitable for very scalable software components, and its preliminary characteristics sketched out. Various directions for future work on the topic have been highlighted throughout the document. However, looking at future many-core platforms embodying Network-on-a-Chip architectures, one of the toughest challenges is the one to incorporate in this kind of models the interferences due to the overlapping paths of the communications among cores, as well as between the cores and the main memory elements.

References

[1] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli. QoS management through adaptive reservations. *Real-Time Systems Journal*, 29(2-3):131–155, March 2005.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[3] T. P. Baker and S. K. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Proc. 21st Euromicro Conf. Real-Time Systems ECRTS '09*, pages 141–150, 2009.

[4] S. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symp RTAS 2004*, pages 536–543, 2004.

[5] S. Baruah and J. Goossens. The edf scheduling of sporadic task systems on uniform multiprocessors. In *Proc. Real-Time Systems Symp*, 2008.

[6] S. Baruah and G. Lipari. Executing aperiodic jobs in a multiprocessor constant-bandwidth server implementation. In *Proc. 16th Euromicro Conf. Real-Time Systems ECRTS 2004*, pages 109–116, 2004.

[7] E. Bini, G. Buttazzo, and M. Bertogna. The multi supply function abstraction for multiprocessors. In *Proc. 15th IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications RTCSA '09*, pages 294–302, 2009.

[8] R. G. Brown. Maximizing beowulf performance. In *in Proc. of the 4th Annual Linux Showcase and Conference*, 2000.

[9] G. Buttazzo, E. Bini, and Yifan Wu. Partitioning real-time applications over multicore reservations. 7(2):302–315, 2011.

[10] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '98, pages 286–, Washington, DC, USA, 1998.

[11] J.-J. Chen, C.-Y Yang, T.-W. Kuo, and C.-S. Shih. Energy-efficient real-time task scheduling in multiprocessor dvs systems. In *Proc. Asia and South Pacific Design Automation Conf. ASP-DAC '07*, 2007.

[12] T. Cucinotta, F. Checconi, L. Abeni, and L. Palopoli. Self-tuning schedulers for legacy real-time applications. In *Proceedings of the 5^{th} European Conference on Computer Systems (Eurosys 2010)*, Paris, France, April 2010. European chapter of the ACM SIGOPS.

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: towards a realistic model of parallel computation. *SIGPLAN Not.*, 28:1–12, July 1993.

[14] A. Davare, Qi Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proc. of DAC'07*, San Diego, California, USA, June 2007.

[15] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 114–118, New York, NY, USA, 1978. ACM.

[16] R. Guerrra and G. Fohler. A gravitational task model for target sensitive real-time applications. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, pages 309 –317, july 2008.

[17] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31:532–533, May 1988.

[18] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[19] K. Konstanteli, T. Cucinotta, and T. Varvarigou. Optimum allocation of distributed service workflows with probabilistic real-time guarantees. *Springer Service Oriented Computing and Applications*, 4(4), Oct 2010.

[20] K. Lakshmanan, D. de Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, dec. 2009.

[21] H. Leontyev, S. Chakraborty, and J. H. Anderson. Multiprocessor extensions to real-time calculus. In *Proc. 30th IEEE Real-Time Systems Symp. RTSS 2009*, pages 410–421, 2009.

[22] X. Li and M. Malek. Analysis of speedup and communication/computation ratio in multiprocessor systems. In *Proc. Real-Time Systems Symp.*, pages 282–288, 1988.

[23] F. Nemati, M. Behnam, and T. Nolte. Independently-developed real-time systems on multi-cores with shared resources. In *Proc. 23rd Euromicro Conf. Real-Time Systems (ECRTS)*, pages 251–261, 2011.

[24] K. Ramamritham, J. A. Stankovic, and P. F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst.*, 1:184–194, April 1990.

[25] Yuan Shi. Reevaluating amdahl's law and gustafson's law. http://www.cis.temple.edu/ shi/docs/amdahl/amdahl.html, October 1990.

[26] A. Srinivasan, P. Holman, J. H. Anderson, and S. Baruah. The case for fair multiprocessor scheduling. In *Proc. Int. Parallel and Distributed Processing Symp*, 2003.

[27] X.-H. Sun and L. M. Ni. Another view on parallel speedup. In *Proc. Supercomputing '90*, pages 324–333, 1990.

[28] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, SODA '94, pages 112–121, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[29] Otilia Werner-Kytölä and Walter F. Tichy. Self-tuning parallelism. In *Proceedings of the 8th International Conference on High-Performance Computing and Networking*, HPCN Europe 2000, pages 300–312, London, UK, 2000. Springer-Verlag.