Multi-Amdahl: Optimal Resource Sharing with Multiple Program Execution Segments

Tsahee Zidenberg, Isaac Keslassy, and Uri Weiser

***Abstract*—This paper presents Multi-Amdahl, a resource allocation analytical tool for heterogeneous systems. Our model includes multiple program execution segments, where each one is accelerated by a specific hardware unit. The acceleration speedup of the specific hardware unit is a function of a limited resource, such as the unit area, power, or energy.**

Using the Lagrange theorem we discover the optimal resource distribution between all specific units. We then illustrate this general Multi-Amdahl technique using several examples of area and power allocation among several cores and accelerators.

I. INTRODUCTION

In the past few years, chip designers have increasingly taken into account resource constraints, most notably power, as a design goal. Their focus has shifted from improving performance to improving performance within a limited power envelope. Heterogeneous cores have been suggested for performance/power ratio improvement. These units are designed for specific workloads, trading efficiency for flexibility. The shift towards special-purpose hardware can be seen in today's CPU products, which add a graphic accelerator to the general-purpose cores [1], [2], [3]. Another example of this trend can be seen inside the general-purpose core; special-purpose logic is added, supporting specific computation, such as CRC or cryptography [4].

In multicore environment it is necessary to correctly balance the performance of parallel and serial code segments to overcome the Amdahl law ceiling [5]. Parallel code runs most efficiently when splitting the available area into many processors, while the serial code can only run on a single processor. The difference between the requirements of the two sections created the *asymmetric-cores* approach [6]. An optimal point for the asymmetry can be found using Amdahl's law for balancing the importance of parallel and serial execution, along with the fact that all processors share a common resource [7].

As hardware becomes more specialized and diverse, Amdahl's law becomes *multidimensional*. In this environment not only some segments are accelerated while others are not, but also different accelerated segments might rely on different types of accelerators. Some of these segments could represent high-level computation sections, such as matrix multiplication and FFT, while others could represent low-level computation sections, such as sections with many floating-point instructions.

All writers are with the department of Electrical Engineering, Technion, Haifa 32000, Israel. As usual, this technical report has not been peer-reviewed and does not count as a publication. It simply presents a few preliminary results.

A limited resource, such as power or area, is shared among the various specific units on the chip. Our target is to find the optimal way to distribute that resource between the specific HW units, balancing the efficiency of different hardware units with their importance and performance.

This paper presents two main contributions:

- We propose Multi-Amdahl, an analytical tool to optimize resource allocation among n different specific HW units running n segments of execution code. The architect may impose constraints on the design, such as total area, power, or energy, and expect optimal outcome, such as maximum speedup. In our model, we take into account the differences in efficiency and scalability of hardware units, and the workload distribution among the different segments.
- Initial results and intuitions obtained from Multi-Amdahl are presented. These results suggest that the opportunities that exist in heterogeneity might surpass the cost of inflexibility. In other words, the occasional use of an accelerator might exceed the costs resulting from its frequent inactivity.

The paper structure is as follows: Section II covers the related work, Section III presents the Multi-Amdahl technique. Our technique is compared with Amdahl's law in Section IV. In Sections V and VI, we present a few examples for extending the basic model. Section VII presents results from our model and their implications, and Section VIII concludes and points to potential future extensions of our work.

II. RELATED WORK

The move towards accelerators

Venkatesh et al. [8] explored the problem of "dark silicon". Today, threshold voltage no longer scales when technology advances. The result is that a shrinking percent of the chip may be activated simultaneously. To achieve peak performance, we must make sure that minimal power is spent for each function, so that more functions could be executed in parallel. The article suggested an architecture with many heterogeneous cores, each designed for optimal power efficiency of a different software function. Those automatically-generated units provided up to 30x efficiency (work/J) over general-purpose MIPS. The article also suggested "patchable" versions of those units with lower (16x) improvements.

Shee et al. [9] tested a few architectures for a heterogeneous chip built especially to encode JPG images. Each core was optimized for one specific pipeline stage. Mostly, optimization was done by removing unnecessary components from a general-purpose core. Hameed et al. [10] created an even more specific processor for video encoding by adding custom-made "magic" instructions. The final chip was about $256\times$ faster than the original RISC processor, while consuming about 1% of the energy and 126% of the area.

Chung et al. [11] compared the power efficiency of general-purpose cores with three forms of "unconventional cores": FPGA, GPGPU and custom logic. Depending on the benchmark, custom logic was shown to be around 100x times as power-efficient (performance ratio / power ratio) as a CPU. The ratio for FPGAs and GPGPUs was, depending on the benchmark, around 10x and 5x respectively.

A large body of work by both academia and industry is dedicated to supporting the heterogeneous compute environment. Various frameworks have been suggested for different aspects of the heterogeneous environment, including programing [12], run-time [13] and hardware [14].

Analytical models for the multiprocessor

Hill and Marty [7] created an initial analytical model revealing the relationship between single thread execution speedup gained from using larger cores, and the multi-threaded phase execution speedup gained from using more cores, under a total budget. The model shows that the optimal results are achieved by asymmetric multicores, with one large core for accelerating the single-threaded phase and many small cores for accelerating the multi-threaded phases.

Woo et al. [15] extended this model for $Performance/Watt$ and $Performance/Joule$ measures. Three systems are modeled: symmetric full-blown processors, symmetric efficient processors, and asymmetric ones. The three systems can be compared with one another by limiting them to an equal power budget. Once again, the asymmetric multicore shows better results for almost any measure.

Chung et al. [11] also extended Hill's model. Their model took into considerations three different budgets: total area (as in [7]), total power (similar to [15]), and total bandwidth. Three models of chips were tested - symmetric multicore, asymmetric multicore, and heterogeneous. The last model is an extension of the asymmetric one, where the efficient cores are in fact "unconventional cores" and therefore even more power efficient. The flexibility of unconventional cores was not modeled. All the unconventional cores in the model can execute the entire parallel portion of the workload. The authors of the article have made a few projections on the future technology nodes and the changes in overall bandwidth budget, and have reached the conclusion that bandwidth, rather than power, would be the main reason for performance limitation in the future.

Our technique is different from the ones presented above, by the fact that we provide optimal solution for n different execution segments rather than only two segments. Multi-Amdahl models the cost of the inflexibility introduced in the heterogeneous system.

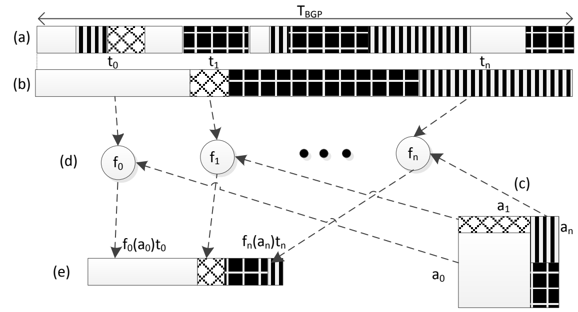

Figure 1. Basic model (a) BGP execution time, (b) aggregated BGP execution time, (c) resource allocation (e.g. area), (d) acceleration function, (e) final execution time

III. Multi-Amdahl

A. Entities

Multi-Amdahl is a strategy for finding the optimal resource assignment for different accelerators sharing common limited resources.

Figure 1 illustrates an example for the three basic entities of the technique:

The Workload — Figure 1(a) presents execution time as measured running on a Basic General-Purpose core (BGP). Figure 1(b) presents execution time when it is aggregated and divided into n segments, each of which will run on a different accelerator. For simplicity, we do not model any cost for moving context between segments. This is common in such models (e.g. [7], [15], [9]). Segment i takes t_i seconds to execute on the BGP. The parameters $t_i, 0 \leq i < n$ represent the workload's distribution between the different execution segments.

$$T_{BGP} = \sum t_i$$

Resource and Constraint — Figure 1(c) illustrates how the chip is divided into n hardware units. The units share a common resource (e.g. area, power, energy). The chip design aims at resource allocation under a specific constraint. For example, when allocating area A to different hardware units, the constraint is the total die area A,

$$\sum x_i \leq A \tag{1}$$

The resource units are normalized so that the BGP uses one unit. Different resources and constraints are presented in Section V.

Efficiency — Figure 1(d) presents unit efficiency, which is a function determining how long section i will take to execute when assigned x_i resource.

$$T_i = t_i f_i(x_i)$$

Each unit may be described by a different function. The function represents the unit's technology. For example, when considering number of transistors (normalized to BGP transistor number) as a resource, the function:

$$f(x) = \frac{1}{100\sqrt{x}}$$

models an accelerator that is 100 times as efficient as a BGP when assigned BGP-equivalent transistors, but it will only double its performance if assigned four times more transistors, according to Pollack's law [16]. Theses functions hide details of how the resource is used (e.g. is the area divided into many narrow units or a few wide ones).

Figure 1(e) illustrates the total aggregated execution time. The optimization goal is to minimize this total time.

$$T_{exec} = \sum T_i = \sum t_i f_i(x_i) \tag{2}$$

Different use-cases for the model are presented in Section VI.

B. Optimization

Lagrange multipliers are a mathematical tool for finding maxima and minima for a multi-dimensional function within a set of constraints on the input variables. In Multi-Amdahl, we minimize total execution output (e.g. time) under the constraint imposed by the resource (e.g. area, power). We assume that additional resources added to the system will create an output gain (e.g. performance), thus the optimal point is inside the limited space where the resource budget is exactly met (i.e., we assume that the Karush-Kuhn-Tucker conditions [17] are met.) Using Equations (1) and (2), the optimization problem can be formalized as:

$$\begin{aligned} \text{minimize} \quad & \sum f_i(x_i) t_i \\ \text{subject to:} \quad & \sum x_i = A \end{aligned}$$

Using the Lagrange optimization method, it follows that the optimal solution satisfies:

$$f_i'(x_i) t_i = f_j'(x_j) t_j \tag{3}$$

The intuition behind Equation (3) is that each infinitesimal additional resource would create the same overall run-time improvement on any accelerator it would be assigned to. This is the basic equation, describing static resources. More complex cases can be analyzed in the same way to provide their optimal point.

IV. Comparing with Amdahl's Law

As an example, we will use our optimization technique to implement a well-known problem of asymmetric processors. This problem is composed out of two execution segments: parallel and serial.

$$t_0 = t_{serial} \quad t_1 = t_{parallel}$$

The parallel section is run most efficiently on many small cores, and the serial on a large core. Therefore, the proposed chip should contain a mixture of both. Hill and Marty [7] analyzed the implications of the chip's limited area. We will use area as a resource (i.e. constraint), and normalize total execution time on the BGP to be 1.

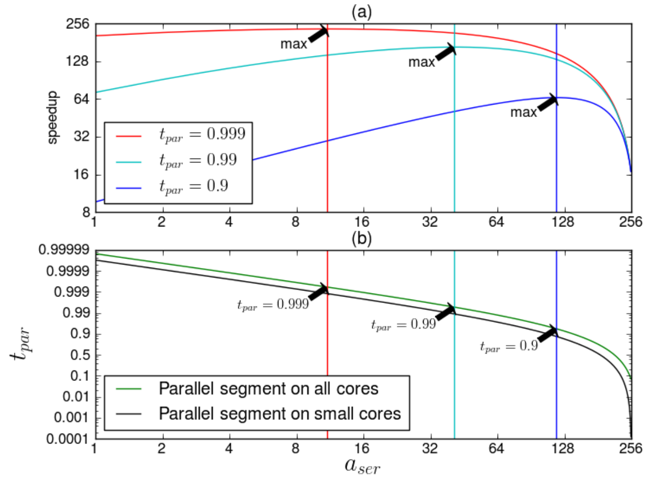

Figure 2. The serial/parallel problem (a) speedup per serial CPU size (according to [7]), (b) optimal serial CPU size per workload

$$\begin{aligned} a_{parallel} + a_{serial} &= A \\ t_{serial} + t_{parallel} &= 1 \end{aligned}$$

The speedup of the parallel section is assumed to be proportional to the number of small cores, and therefore also to the total area of the small cores, which execute the parallel tasks.

$$f_{parallel}(a) = \frac{1}{a}$$

The "serial accelerator" is the large CPU, whose performance scales with area according to Pollack's Law [16]. Formalizing this in our model's terms:

$$f_{serial}(a) = \frac{1}{\sqrt{a}}$$

Applying Equation (3) reveals the optimal relation between the total area of the efficient parallel processors and the area of the serial processor.

$$a_{parallel} = a_{serial}^{3/4} \sqrt{\frac{2t_{parallel}}{1 - t_{parallel}}} \tag{4}$$

An immediate result of our model is the optimal resource allocation point. The speedup obtained at this point could be calculated by Equation (2). The focus of this model allows for a different set of insights. For example, it is apparent from Equation (4) that the serial section grows faster than the parallel one when the chip receives additional resource.

Note that for simplicity we presented a model in which the large core will not be used for execution of the parallel segment. We will use the assumption that different hardware units do not execute the same code in following sections, when expanding the model for many accelerators. On the contrary, Hill and Marty [7] used a different assumption, in which the parallel segment is executed by all the cores on the chip, including the large core. This slightly changes the results, but the Multi-Amdahl optimization technique does not change significantly.

Figure 2 illustrates the optimization results. Figure 2(a), essentially taken from Hill and Marty's paper [7], reveals the existence of optimal resource allocations (a_{serial}) which changes according to the workload ($t_{parallel}$). Figure 2(b) presents the exact value of these optimal resource allocations using Multi-Amdahl technique, and in particular the relation between $t_{parallel}$ and the optimal value of a_{serial}. Results are presented for both cases, where the parallel section is executed either only on the efficient cores or on the entire chip.

V. Different resources

The Multi-Amdahl technique can be applied to different resource types with various constraints.

A. Static resources

A static resource is used by an accelerator for the entire life-time of the problem. For example it could be the die area or the number of transistors. If all hardware units are working concurrently, power and IO might also be modeled as static resources.

When allocating a static resource, the designer's goal is to stay within a total budget X:

$$\sum_{i=0}^{n-1} x_i \leq X$$

B. The Power Resource

Today, the resource-allocation efforts of the chip designer have been shifted to power constraints. When modeling power, we must take into account both *dynamic power*, which is only consumed when the unit itself works, and *static power*, which is also consumed when it is idle.

The actual resource assigned by the designer, however, is still the number of transistors. Both the static and the dynamic power can be modeled as proportional to the number of transistors in the accelerator, so we can model them as linearly dependent. A linear relation between static and dynamic power is a common model, used e.g. by [15]. In our model, each unit is assigned p_i power when in use, and consumes additional $k_i p_i$ static power all the time. k_i is assumed to be another accelerator-technology dependent parameter, known to the chip manufacturer.

Several constraints can be considered for the power resource:

Instantaneous power — There is a total power budget that the multiprocessor may use at any given instant. The budget usually derives from power dissipation consideration. The constraint is imposed on each unit (or execution segment) separately. This constraint is most applicable in case the different segments last long enough to overheat the chip.

$$\forall 0 \leq i < n : \quad p_i + \sum_{j=0}^{n-1} k_j p_j \leq P$$

Energy — The energy represents the total power consumed by the chip over time. It is a design goal for servers, where electricity costs are considerable, and for mobile devices, where minimizing energy consumption is necessary to maximize battery life.

$$\sum k_j p_j \sum f_i(p_i) t_i + \sum f_i(p_i) t_i p_i \leq E$$

Total Dynamic Power — The total (or average) dynamic power is calculated by dividing the overall energy by the overall execution time. If execution segments are short enough, power dissipation poses a constraint on the average power consumption, rather than on the instantaneous one.

$$\sum k_i p_i + \frac{\sum f_i(p_i) t_i p_i}{\sum f_i(p_i) t_i} \leq TDP$$

C. Multiple resources

More than one constraint and resource can be tested at the same time. For example, we are going to discuss the combined effect of assigning supply voltage (marked v_i) and area (a_i).

The maximum operation frequency is proportional to voltage.

$$freq_i = v_i$$

Performance is modeled as linearly proportional to frequency, and sub-linearly proportional to area:

$$f_i(a_i, v_i) = \frac{1}{freq_i \sqrt{a_i}} = \frac{1}{v_i \sqrt{a_i}}$$

Energy is modeled as proportional to area, voltage, and operation time:

$$E_i = f_i(a_i, v_i) t_i a_i v_i^3$$

We have two constraints: one for total area, and one for total energy.

$$\sum a_i \leq A$$

$$\sum f_i(a_i, v_i) t_i a_i v_i^3 \leq E$$

Note that no constraints are directly applied to the voltage.

VI. Different use-cases

Multi-Amdahl could be used to describe different use cases. A use case determines the workload and efficiency functions.

A. Serial Execution

One case of introducing accelerators into a system is when little or no parallelism can be extracted from the code. In this simplified model, we assume the entire chip only executes one segment at a time, and only on the appropriate accelerator. The total execution time is given by:

$$T_{exec} = \sum T_i \quad T_i = t_i f_i(x_i)$$

This was the use-case of all the previous sections.

B. Parallel Execution

In this model, the various accelerators handle a different type of parallel input each. We try to minimize the average latency, given by:

$$T_{latency} = \sum \lambda_i f_i(x_i)$$

λ_i is the rate (inputs per second) for this type of input.

f_i is the latency of calculation of type i when assigned x_i resource.

This model can be applied, e.g. for network processors, where different accelerators handle different types of packages (such as encrypted, compacted..).

C. Optimizing for different units in a CPU

Even inside a basic CPU there are various separate hardware units, some of which can be described as handling their own instruction set. For example, we might consider allocating resource optimally between 3 units: the cache, the branch predictor, and the ALU.

$$CPI = \lambda_c c_c(x_c) + \lambda_p c_p(x_p) + \lambda_a c_a(x_a)$$

λ_c is the number of memory accesses per instructions. In our model, memory accesses are executed by the cache. As the cache is assigned more resource, it becomes larger and the cache hit ratio increases.

$$c_c(x) = hit\%(x) * T_{hit} + (1 - hit\%(x)) * T_{miss}$$

λ_p is the number of branches per instruction. Branches are modeled as executed by the branch predictor. As the cache predictor is assigned more resource, it should improve branch prediction rates.

$$c_p(x) = (1 - predict\%(x)) * T_{mispredict}$$

λ_a is the number of ALU instructions. As ALU is assigned more resource, more ALUs are added to the system which increases throughput of ALU instructions.

$$c_a(x) = \frac{T_{ALU}}{x}$$

VII. Initial Results

Initial results were calculated for a static resource, such as area, and for the serial execution model. Our results indicate that good accelerator efficiency can be put to use even for the price of flexibility. With dynamic resource, the cost of inflexibility is much smaller, and in that sense our results are conservative.

A. The general case

When allocating resources for different accelerators, we must take two elements into consideration: how efficient the accelerator is and how useful it is (meaning, what is its part in the workload). We consider a general efficiency function:

$$f_i(a) = \frac{1}{\alpha_i a^{\beta_i}}$$

A general-purpose CPU can be modeled by $\alpha_0 = 1$. Higher values of α are assigned to more efficient accelerators. We use Multi-Amdahl to extract the appropriate area allocation for each of the accelerators:

$$a_i = a_0^{\frac{\beta_j+1}{\beta_i+1}} \left(\frac{\alpha_0/\beta_0}{\alpha_i/\beta_i} \frac{t_i}{t_0} \right)^{\frac{1}{\beta_i+1}} \tag{5}$$

The most interesting thing this solution reveals is that the workload-dependent parameters (t_i) have equal or lower importance to the parameters that are dependent on the accelerator's technology (α_i, β_i) when determining the optimal solution. This has an implication on the chip manufacturer's ability to allocate resources properly with even a partial knowledge of the workload, which will be analyzed later.

B. Effective heterogeneous speedup

A heterogeneous system might consist of various units, when each can accelerate its designated code segment with noticeable speedup over a general-purpose machine. The program is composed of various segments. The effective speedup is measured over the entire execution, including the general-purpose section, and is generally lower then the speedup for a single section. Multi-Amdahl reveals another effect of heterogeneity. As more heterogeneity is added to the system, the resource is shared between more accelerators, and therefore each accelerator is assigned less resource, thus reducing its speedup.

To display this effect, we consider a system composed of one general-purpose section and n accelerators (notice this system has $n + 1$ segments). All code, accelerated or not, is assumed to be entirely parallelisable ($\beta_i = 1$). All accelerators are equally efficient ($\alpha_{i>0} = \alpha$).

$$f_0(a) = \frac{1}{a} \quad f_{i>0}(a) = \frac{1}{\alpha a}$$

We mark δ to be the part of the original code using the accelerators. We assume this part is equally distributed among the different accelerated segments.

$$t_0 = 1 - \delta \quad t_{i>0} = \frac{\delta}{n}$$

Putting this into Equation (5):

$$a_i = a_0 \sqrt{\frac{\delta}{\alpha n(1-\delta)}}$$

from which we can also derive the total execution time for a chip with area budget A, using Equation (2):

$$T_{het} = \frac{1}{A} \left(2\sqrt{\frac{n}{\alpha}\delta(1-\delta)} + 1 - \delta \left(1 - \frac{n}{\alpha} \right) \right)$$

A homogeneous multicore system uses the entire available area for general-purpose CPU, and executes the entire code without any speedup:

$$T_{hom} = \frac{1}{A}$$

Therefore, the speedup from introducing heterogeneity into the system is:

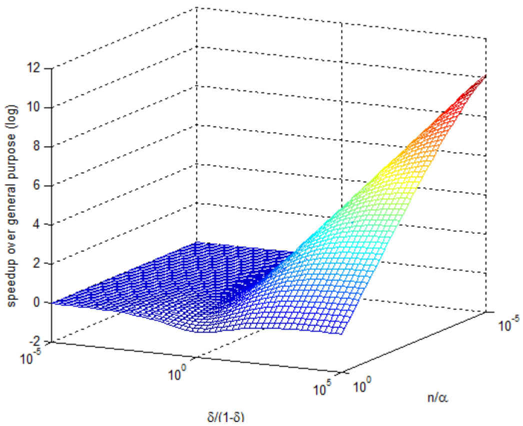

Figure 3. Combined effect of $\frac{n}{\alpha}$ and δ on speedup

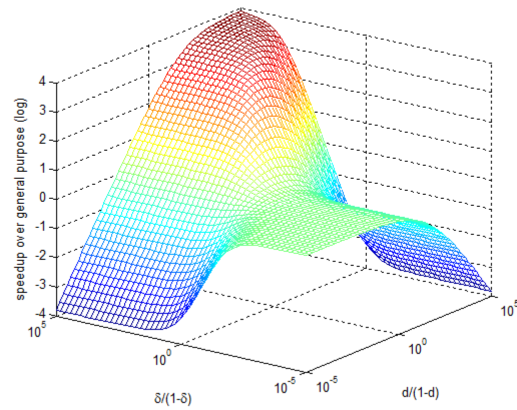

Figure 4. Sensitivity ($\frac{n}{\alpha} = \frac{1}{50}$)

$$Speedup_{het} = \frac{T_{hom}}{T_{het}} = \left(2\sqrt{\frac{n}{\alpha}\delta(1-\delta)} + 1 - \delta\left(1 - \frac{n}{\alpha}\right)\right)^{-1}$$

Speedup vs. Flexibility — The value of n has a tremendous influence on the speedup gained. This is the effect of accelerator's inflexibility. Accelerators use system resources all the time, but they are seldom used for actual computation. The more accelerators are in the multiprocessor, the more "dead area" it contains, per execution segment.

One reason for adding accelerators into a system would be to create more specific, and therefore more efficient, accelerators. The equations present a linear relation between n and α, which fits intuition. An accelerator capable of two operations should be split into two accelerators capable of one operation each, only if those two accelerators are at least twice as efficient as the previous one.

Speedup vs. Code coverage — Another reason for adding accelerators to an existing system would be moving part of the code that previously ran on the general processor to an accelerator, namely to increase δ.

Figure 3 reveals that for low values of δ there is little effect either way. As δ approaches 1, the rule of thumb is that multiplying the number of accelerators is worthwhile if it does better than halve the amount of code not running on accelerators $(1-\delta)$.

C. Resource allocation sensitivity

As we have previously mentioned, the chip manufacturer is unaware of the exact nature of the workload running on the machine, and can only estimate the expected workloads. For that reason we also model the case where the manufacturer creates a chip for a given workload, while the actual workload is different.

In our case, the chip manufacturer assumes equal use of all accelerators, and assumes use of accelerated code $\delta = d$. According to these assumptions, the manufacturer divides the area between the accelerators.

$$a_i = a_0\sqrt{\frac{d}{\alpha n(1-d)}}$$

However, the actual value of δ is different. The speedup of the chip for the actual workload will be:

$$Speedup = \left(\left(1 + \frac{\delta}{d} - 2\delta\right)\sqrt{\frac{n}{\alpha}\frac{d}{(1-d)}} + 1 - \delta\left(1 - \frac{n}{\alpha}\right)\right)^{-1}$$

Figure 4 presents a few properties of the equation: There is no reason for the chip manufacturer to assume small values for d (significantly smaller then 0.5). For those values, high accelerator use will incur a serious slowdown, as most execution time is spent on an accelerator with insufficient resource, while low accelerator use results in minimal speedup, because most of the execution is on the CPU. Large values of d (close to 1) might prove very beneficial for enough accelerator usage, but will be destructive if the accelerators are not used, as the CPU is very weak. Using 0.5, or somewhat larger values for d is the "safe" choice. An observable speedup can be seen when the accelerator is used, while the slowdown for workloads not using the accelerator is negligible.

VIII. CONCLUSION AND FUTURE WORK

This paper presents Multi-Amdahl, an analytical technique for optimal resource allocation in a heterogeneous chip. Our technique relies on the modeling of the resource, the workload, and the accelerators' performance as a function of the chip's resource. We have shown the technique's applicability to a large field of problems. For example, the accelerators considered may either be part of the general-purpose cores or separate accelerators.

We have used our model to test the importance of accelerator efficiency vs. code coverage, and have found the two parameters to be equal when looking for the optimal resource allocation. We have also discussed the case of workload variance, and found a "sweet-spot" for chip design, characterized by minimal slowdown when the accelerator is not used, versus a measurable speedup when the accelerator is used. Those results are based on an environment in which moving context between accelerators has no overhead, and resource allocation is static.

Generally speaking, our results suggest that inflexibility is a reasonable price to pay for efficiency, and that accelerator-based heterogeneous multicores are a promising direction for future chip architectures.

Our future research will concentrate on the expansion of the applications area, while putting more overhead and resource allocations constrains.

ACKNOWLEDGMENT

This research work was partly supported by an Intel research grant on heterogeneous computing, by the European Research Council Starting Grant n° 210389, by the Hasso Plattner Center for Scalable Computing, and by the Israeli MOST CMP Research Center.

REFERENCES

[1] H. Jiang and T. A. Piazza, "Intel next generation microarchitecture code named sandybridge," in *Intel Developer Forum*, 2010.

[2] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4.5, pp. 589–604, july 2005.

[3] N. Brookwood, "Amd fusion family of apus," *Insight 64*, 2010.

[4] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund *et al.*, "Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu," in *ISCA '10: Proceedings of the 37th annual International Iymposium on Computer Architecture*. ACM, 2010, pp. 451–460.

[5] 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*. ACM, 1967, pp. 483–485.

[6] T. Y. Morad, U. C. Weiser, and A. Kolodny, *"ACCMP-Asymmetric Cluster Chip Multi-Processing"*. Citeseer, 2004.

[7] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[8] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: reducing the energy of mature computations," *ACM SIGPLAN Notices*, vol. 45, no. 3, pp. 205–218, 2010.

[9] S. L. Shee, A. Erdos, and S. Parameswaran, "Architectural exploration of heterogeneous multiprocessor systems for jpeg," *International Journal of Parallel Programming*, vol. 36, no. 1, p. 140, 2008.

[10] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *ISCA '10: Proceedings of the 37th annual International Symposium on Computer Architecture*. ACM, 2010, pp. 37–47.

[11] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus?" in *MICRO-43: Proceedings of the 43th Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[12] A. Munshi, "Opencl," *Parallel Computing on the GPU and CPU, SIGGRAPH*, 2008.

[13] C. Augonnet, S. Thibault, R. Namyst, and P. A. Wacrenier, "Starpu: A unified platform for task scheduling on heterogeneous multicore architectures," *Euro-Par 2009 Parallel Processing*, pp. 863–874, 2009.

[14] M. Lyons, M. Hempstead, G. Y. Wei, and D. Brooks, "The accelerator store framework for high-performance, low-power accelerator-based systems," *Computer Architecture Letters*, vol. 9, no. 2, pp. 53–56, 2010.

[15] D. H. Woo and H. H. S. Lee, "Extending amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, no. 12, pp. 24–31, 2008.

[16] 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.

[17] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability. July 31-August 12, 1950.*, vol. 1, 1951, pp. 481–492.