Mind The Power Holes: Sifting Operating Points in Power-Limited Heterogeneous Multicores

Almutaz Adileh, Stijn Eyerman,
Aamer Jaleel, and Lieven Eeckhout

Abstract—Heterogeneous chip multicore processors (HCMPs) equipped with multiple voltage-frequency (V-F) operating points provide a wide spectrum of power-performance tradeoff opportunities. This work targets the performance of HCMPs under a power cap. We show that for any performance optimization technique to work under power constraints, the default set of V-F operating points in HCMPs must be first filtered based on the application's power and performance characteristics. Attempting to find operating points of maximum performance by naively walking the default set of operating points leads the application to inefficient operating points which drain power without significant performance benefit. We call these points *Power Holes (PH)*. Contrary to intuition, we show that even using a power-performance curve of Pareto-optimal operating points still degrades performance significantly for the same reason. We propose PH-Sifter, a fast and scalable technique that sifts the default set of operating points and eliminates power holes. We show significant performance improvement of PH-Sifter compared to Pareto sifting for three use cases: (i) maximizing performance for a single application, (ii) maximizing system throughput for multi-programmed workloads, and (iii) maximizing performance of a system in which a fraction of the power budget is reserved for a high-priority application. Our results show performance improvements of 13, 27, and 28 percent on average that reach up to 52, 91 percent, and 2.3×, respectively, for the three use cases.

Index Terms—Heterogeneous multicores, power-limited processors, optimal operating points, power management

1 Introduction

Heterogeneous multicore processors (HCMPs), e.g., ARM big.LITTLE, have the potential to expand the set of power-performance tradeoff points in CMPs beyond DVFS. However, this added flexibility significantly complicates mining for optimal operating points that achieve the power and performance targets. In this paper, we seek to optimize the performance of power-limited HCMPs.

A power limit entails a period of time during which the power rate must be maintained. For example, thermal design power (TDP) is maintained over a thermally-significant period: instantaneous power can exceed TDP at times, as long as TDP is maintained within that period [1]. To maximize performance of one application on one core type, a traditional power manager uses the default V-F operating points to generate a performance-power curve for the application. Then it walks the curve to find the point of maximum performance that does not exceed the limit. A power target in between two points can be achieved by alternating between the points. The same approach applied to power-limited HCMPs leads to suboptimal performance.

We observe that when mixing different core types in HCMPs, naively relying on the default set of operating points leads to sub-optimal results. Finding the highest performing point below the power limit, and alternating with another point (e.g., immediately) above the limit can waste significant power and performance. We show that, contrary to intuition, even when filtering the default set of operating points to keep only Pareto optimal points, this approach still leads to significant performance degradation. Using a brute-force approach to find the points of maximum performance involves high overhead. This is especially true with a high number of core types, operating points per core, and concurrent applications. In particular, optimizing performance of multiple applications sharing a power budget explodes the search space as it requires both optimal budget division among applications and optimal per-application operating point selection based on the assigned budget.

In this paper we show that several points in the default set of operating points drain power without significant performance benefit, whenever they are used in power-limited HCMPs; hence, the name *power holes*. Surprisingly, Pareto-optimal points suffer a similar problem. Sifting power holes is key to optimizing performance for both cases of single and multiple concurrent applications. Moreover, the sifting must be performed at runtime as it is application-specific. We propose PH-Sifter, a fast and scalable technique for sifting operating points, and keeping only the set of points that optimally use power to gain performance. We show significant performance improvement of PH-Sifter compared to Pareto sifting for three use cases: (i) maximizing performance for a single application, (ii) maximizing system throughput for multi-programmed workloads, and (iii) maximizing performance of a system in which part of the power budget is reserved for a high-priority application. Our results show performance improvements of 13, 27, and 28 percent on average that reach up to 52, 91 percent, and 2.3×, respectively, for the three use cases.

2 Background And Motivation

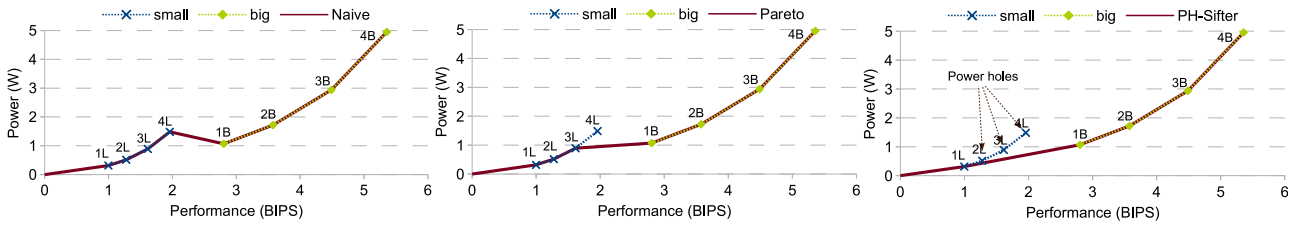
Our work targets the performance of power-limited HCMPs. Because the V-F operating points come in discrete values, power managers need to search for the highest performing point that does not violate the power limit [2]. Conservatively selecting a single operating point does not guarantee maximum performance under a power limit. First, the difference between that point's power rating and the allowed budget is wasted, degrading performance. More importantly, higher performance could be achieved within the same limit when switching the application between two points; one below the power limit and one that exceeds the limit but has higher performance. One approach to find the optimal operating points is to use a brute force search for all the pairs of operating points. However, this method suffers a significant overhead for a single application, and the overhead explodes as the number of operating points and concurrent applications increases.

Fig. 1 shows the performance versus power consumption for two core types, each with four operating points, for an example benchmark, bzip2.liberty from SPEC CPU2006. Each point represents one V-F setting of the core (see Section 4 for details regarding the experimental setup). Fig. 1 shows the naive approach to maximize performance under a power limit. It starts by walking the default operating points from the lowest performance point on the little core to the highest point that does not exceed the power limit. Then it opportunistically alternates between this point and the next higher performance point, to leverage the whole power budget, while keeping the average power below the limit.

Assuming a power limit of 1 W for example, this naive approach schedules the application on the third point of the little core (3L) and switches to the fourth point (4L) as much as the budget allows. This approach results in sub-optimal performance

- *A. Adileh and L. Eeckhout are with Ghent University, Gent, East Flanders 9052, Belgium. E-mail: almutaz.adileh@ugent.be, lieven.eeckhout@elis.ugent.be.*
- *S. Eyerman is with Intel Belgium, Leuven, Kontich 2550, Belgium. E-mail: stijn.eyerman@elis.ugent.be.*
- *A. Jaleel is with Nvidia Research, Boston, MA 01886. E-mail: ajaleel@nvidia.com.*

Manuscript received 3 May 2016; revised 20 Sept. 2016; accepted 28 Sept. 2016. Date of publication 10 Oct. 2016; date of current version 26 June 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2016.2616339



(a) Naive performance-power curve walking (b) Pareto sifting (c) PH-Sifter

Fig. 1. Performance-power curves considering four operating points for the big and little cores.

because alternating between point 3L of the little core and 1B of the big core better trades power for performance. A method that considers only Pareto-optimal performance-power points filters point 4L, as Fig. 1b shows.

An operating point is considered Pareto-optimal if there exists no other operating points that yields better performance at lower power. Point 4L of the little core is a non-Pareto-optimal point, and can therefore be discarded. All other operating points are Pareto-optimal. Pareto curves are usually used to show the spectrum of efficient operating points in HCMP products [3]. Surprisingly, walking the Pareto-optimal curve to find the optimal operating points within the power budget [4] still leads to power holes that waste power and degrade performance.

Consider the same application and the same 1 W power budget. An approach that uses Pareto-optimal points alternates between points 3L (little) and point 1B (big). Assuming no migration overhead in this example, the application runs 62 percent of the time on the big core and 38 percent on the little core, reaching an average performance of 2.35 BIPS. The Pareto approach improves over the naive because the line between points 3L and 1B crosses the 1 W budget line further to the right on the *x*-axis. However, there are still opportunities to improve performance. For example, point 2L uses less power than 3L and allows the application to utilize the big core at point (1B) up to 88 percent of the time, for a higher performance of 2.62 BIPS. As we show in Section 3, this latter choice is still not the optimal. This calls for a feasible approach to identify optimal operating points that maximize performance under power limits.

3 PH-Sifter

Based on the discussion in Section 2, the problem of filtering operating points reduces to selecting the next point that gives the highest performance for power, starting from a lower operating point. In this section, we propose PH-Sifter, a fast operating point sifting technique. PH-Sifter relies on Delta Performance/Delta Power (DP/DP) to rank the relative efficiency of optimal operating points. DP/DP is the reciprocal of the slope between two operating points. Hence, the flatter the slope, the higher DP/DP. In Fig. 1, the slope between the third and fourth points of the little core (3L versus 4L), is steeper than the slope between the third point of the little core and the first point of the big core (3L versus 1B).

Now consider the line between the lowest operating point of the little core and the lowest point of the big core as in Fig. 1c. Its slope is lower (i.e., its DP/DP is higher) than all other lines originating from the lowest operating point of the little core to any other operating point. Intuitively, alternating an application between the lowest point of the little core (1L) and the lowest point of the big core (1B) results in a (virtual) operating point on this line. It is clear that any other operating point on the little core results in a worse power/performance (virtual) operating point. As a result, all operating points of the little core can be pruned except for the lowest point.

A set of operating points can be proven optimal for performance optimization under a power limit *if (and only if)* DP/DP is a monotonically decreasing function from the lowest to the highest power-performance operating point. Fig. 1 visualizes this. The Pareto frontier in Fig. 1b is not a monotonically decreasing function: the slope between the second and third points of the little core (3L versus 4L) is steeper than between the third point of the little core and the first point of the big core (3L versus 1B).

Put differently, the monotonicity of the DP/DP metric means that the curve of optimal operating points should be convex, i.e., no point should be above any line connecting two other points. For the power manager, this means that the *next operating point to be considered is the one with the highest DP/DP relative to the current operating point, and all intermediate operating points are* power holes *that should be pruned.*

3.1 Algorithm

Algorithm 1 describes PH-Sifter. The algorithm starts at the lowest operating point of the little core. It then chooses the next optimal point as the one with the highest DP/DP value. All the intermediate points between the two selected points are considered power holes and are filtered out. From the newly chosen point, the algorithm again selects the point with the highest DP/DP value from the set of operating points that were not filtered out already. The algorithm proceeds until it reaches the highest operating point of the big core. Note that Algorithm 1 applies to HCMPs with more than two different core types.

Algorithm 1. PH-Sifter: Excluding Power Hole Operating Points

$init\_points$ = all operating points form all core types
$sifted\_points$ = NULL
Sort (ascending) $init\_points$ by performance values
$current\_point = init\_points[0]$
Push $init\_points[0]$ to $sifted\_points$, pop it from $init\_points$
while $current\_point$ is not last element in $init\_points$ **do**
  Calculate DP/DP from $current\_point$ to all $init\_points$
  Take highest DP/DP as $highest\_point$
  Remove all intermediate points between $current\_point$ and $highest\_point$ from $init\_points$
  Push $highest\_point$ to $sifted\_points$, pop it from $init\_points$
  $current\_point = highest\_point$
end while
output $sifted\_points$

Excluding intermediate points at each step of Algorithm 1 results in the convex shape that guarantees optimality of the sifted set. For any potential budget, the two points in the set with power values just above and below the budget, form a line that crosses the budget line furthest to the right (i.e., highest in performance).

Consider Fig. 1c as an example. Starting from point zero, point 1L has the highest DP/DP. Index $current\_point$ is set to point 1L and it is included in the optimal set. Next, from 1L, point 1B has the highest DP/DP, so it is included in the optimal set, $current\_point$ is set to 1B, and intermediate points (2L, 3L, 4L) are filtered. Repeat until $current\_point$ reaches 4B.

3.2 Multiple Concurrent Applications

As the number of available cores and concurrent applications increases, operating point sifting becomes more important.

TABLE 1
Voltage-Frequency Settings Used in the Experiments

Frequency (GHz)	Voltage (V)
1.5	0.64
2.0	0.74
2.6	0.88
3.2	1.1

Algorithm 2 shows how to maximize the performance for multiple concurrent applications. In principle, the algorithm walks the power-performance curves for all the applications at the same time starting with the lowest operating point on the little core. At each step, the algorithm selects the application with the highest DP/DP value and provides it with the budget necessary to move to the next higher operating point. The algorithm updates the new operating point of that application and repeats the procedure until the power budget is distributed among the applications.

Algorithm 2. Maximizing Throughput of Multiple Applications

Sift each application (PH-Sifter) to get $sifted\_points$ per app.
$current\_point[App_i] = sifted\_points[App_i][0]$
$next\_point[App_i] = sifted\_points[App_i][1]$
$used\_budget = \sum$(power of all apps on $current\_point$)
while $used\_budget < available\_budget$ **do**
  Calculate DP/DP per app between $current\_point$ & $next\_point$
  Sort the results & take the highest ranked application a
  if $available\_budget - used\_budget \geq next\_point[a].power - current\_point[a].power$ **then**
    Schedule application a on next_point[a]
    $used\_budget = used\_budget + next\_point[a].power - current\_point[a].power$
    $current\_point[a] = next\_point[a]$
    $next\_point[a] = next\_point[a] + 1$
  else
    $fraction_{next\_point}(a) = \frac{available\_budget - used\_budget}{next\_point[a].power - current\_point[a].power}$
    Budget fully consumed, end while loop
  end if
end while
Schedule the last application on $next\_point$ for $fraction_{next\_point}$ and on $current\_point$ for the rest of the time
Schedule the rest on their $current\_point$

As system throughput sums the performance gain achieved by each application, each step in Algorithm 2 allocates power to the application with the highest potential gain, and consequently the highest impact on system throughput. Allocating part of the budget to another application at any step lowers throughput. The optimality of both algorithms can be mathematically proven.

Note that Algorithm 2 would fail to maximize performance without properly-sifted operating points. *Without PH-Sifter, a power manager may have to resort to an expensive non-scalable brute-force approach to optimally divide the power budget and guarantee maximum performance for this case.* For n applications, Algorithm 2 has a complexity of $O(n)$ because it needs to find the application with the highest DP/DP at each step. The total number of steps depend on the power budget.

4 Experimental Setup

We use Sniper 6.0 [5] to perform experiments in this paper. We simulate HCMPs consisting of two core types, big and little. The big core is aggressive four-wide out-of-order, while the little core is two-wide in-order. We assume an LLC of 4 MB per pair of big and little cores that is shared by all the cores. We consider the operating points shown in Table 1.

We use all SPEC CPU2006 benchmarks using all the reference inputs. For multi-programmed workloads, we randomly generate 75 mixes of four applications and run each mix on a processor with four big/little pairs. We use McPAT 1.3 [6] to estimate power consumption assuming 22nm chip technology and idle cores are power-gated. We assume 20 μs [7] core migration overhead, and account for all related overheads (e.g., cache warmup).

We simulate applications for one second of execution. The simulated power manager starts by profiling applications on the middle V-F operating point of each core type for 4 ms. It then estimates the power and performance of the remaining operating points using models from [8], sifts the points and distributes the budget. To adapt to changes in program behavior, the manager gathers power and performance statistics at runtime, regenerates curves, and repeats the same process every 100 ms.

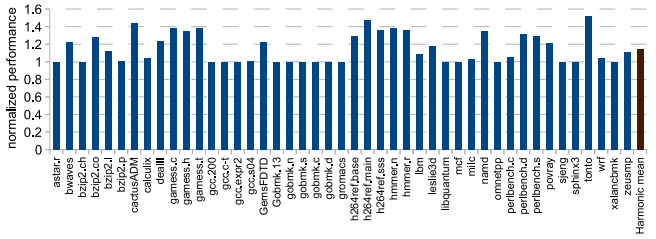


Fig. 2. PH-Sifter performance gain over Pareto sifting for a single application with four points per core type.

5 Evaluation

Operating point sifting is necessary for any optimization targeting the performance of power-limited HCMPs. We evaluate PH-Sifter under three scenarios. The first case shows the performance improvement of a single application using PH-Sifter. The second case extends the first to multiple concurrent applications. Finally, we look into a case with one high-priority application that requires a fixed fraction of the power budget, leaving the remaining budget to other lower-priority applications.

5.1 Maximizing Performance of a Single Task

Fig. 2 shows the performance gain of PH-Sifter normalized to Pareto sifting for the SPEC CPU 2006 applications. Applications run on a pair of big/little cores each with four operating points, assuming a power budget of 1W. The results show an average performance improvement of 13 percent, up to a maximum of 52 percent. The average is affected by applications that do not benefit from pruning under the 1 W budget assumption. For these applications, the budget falls between optimal points that exist in both PH-Sifter and Pareto sifting. If the budget for these applications changes, due to sharing the budget with other applications or due to a fluctuation in the power consumption of other components (e.g., GPUs), PH-Sifter will similarly show high performance benefits.

5.2 Maximizing Performance of Concurrent Tasks

We use Algorithm 2 to maximize the performance for multi-programmed workloads. Fig. 3 shows the performance gain of PH-Sifter over Pareto sifting with 2, 4, 8 and 16 operating points per core type. Each point on the x-axis represents a mix of four applications, for a total of 75 mixes. The workloads are sorted based on their normalized system throughput (STP, on y-axis) [9].

The figure shows that PH-Sifter outperforms Pareto sifting for different numbers of operating points. This shows the impact of power holes on Pareto sifting performance. Additionally, we notice

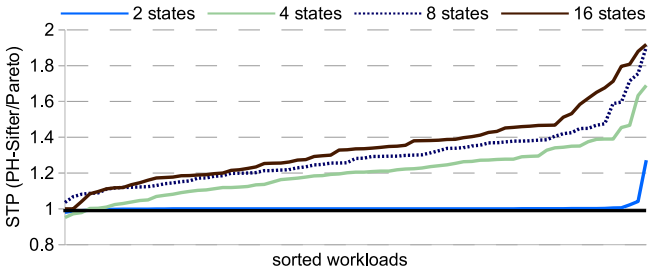


Fig. 3. Comparing PH-Sifter vs Pareto sifting for different numbers of points per core type, using multiprogrammed workloads.

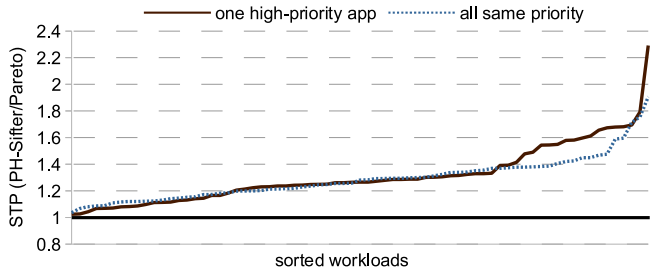


Fig. 4. Performance gain of PH-Sifter over Pareto sifting while provisioning for a high-priority application.

higher improvement as the number of operating points increases. As the number of operating points increases, more power holes need to be sifted. We notice that Pareto sifting fails to identify a larger number of non-optimal points causing the gap with PH-Sifter to widen. The lowest gains are for two points per core, with only a few workloads showing noticeable gains. For these experiments, we use only the lowest and highest points per core type. We find that Pareto sifting in most cases correctly identifies the high point on the little core as a power hole, similar to PH-Sifter (Fig. 1, considering only the lowest and highest points).

Our results show that proper sifting of operating points significantly impacts performance. An average performance gain of 19, 27, and 32 percent is achieved over Pareto sifting, that reaches up to 69, 91, and 92 percent for 4, 8, and 16 points per core, respectively.

5.3 Provisioning for QoS

In this section we extend over the case of multiple concurrent applications to show that proper sifting is required for the general case of improving the performance of power-limited HCMPs.

We consider a scenario where a fixed fraction of the power budget is reserved solely for a high-priority application to meet its performance target, while the remaining budget is divided among the remaining lower-priority applications. This scenario could be further generalized to one where the high-priority application consumes a variable fraction of the power budget, while the remaining budget goes to the other applications.

Our experiment assumes 25 percent of the power budget is allocated to the high-priority application. Fig. 4 shows the results of PH-Sifter compared to Pareto sifting assuming eight operating points per core type. The figure shows again the result of the case without a high-priority application, to show the similarity. Although the same power budget is allocated to the latency-sensitive application, PH-Sifter significantly improves system performance by an average of 28 percent and up to 2.3× compared to Pareto sifting.

This case could be viewed as a mix of the two previously shown use cases. The application risks not meeting its quality target using only 25 percent of the power budget if the operating points are not properly sifted. Similarly, the maximum system performance of the remaining applications would suffer significantly if the power holes are used per application.

6 Conclusion

This paper demonstrates the necessity to properly sift the default set of operating points when optimizing for performance in power-limited HCMPs. We show that naively walking the power-performance curve of the default operating points leads to power holes, or points that waste power for low performance benefit. Selecting operating points using Pareto-optimal sifting suffers similar performance degradation. We propose PH-Sifter, a fast operating point sifting technique that starts at the lowest operating point, filters all the points until the next operating point that achieves the highest Delta Performance/Delta Power value, and continues until all the points are considered. We reason about the optimality of PH-Sifter and show significant performance improvements compared to Pareto sifting for a single application, multiple applications, and systems with high-priority applications.

Acknowledgments

We thank the anonymous reviewers for their thoughtful feedback. This research is supported in part through the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 259295. This work was done while Stijn Eyerman was at Ghent University.

References

[1] A. Raghavan, et al., "Computational sprinting," in *Proc. 18th Intl. Symp. High Performance Comput. Archit.*, 2012, pp. 249–260.
[2] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang, "PPEP: Online performance, power, and energy prediction framework and DVFS space exploration," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 445–457.
[3] MediaTek, "Corepilot 3.0 max.mid.min (tri-cluster) technology to maximize power efficiency with extreme computing performance," MediaTek White paper, 2015.
[4] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 26–36.
[5] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 3, p. 28, 2014.
[6] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Int. Symp. Microarchitecture*, 2009, pp. 469–480.
[7] P. Greenhalgh, "big.LITTLE processing with ARM Cortex-A15 & Cortex-A7," ARM White paper, Sep. 2011.
[8] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting performance impact of DVFS for realistic memory systems," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2012, pp. 155–165.
[9] S. Eyerman, and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May 2008.