Heterogeneous Multi-core Architectures: Optimizing Power and Performance

Naman Jain, Prashanth Suresh
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh PA 15213
{namanj, psuresh}@andrew.cmu.edu
Project URL: http://www.contrib.andrew.cmu.edu/~namanj/18-743/

***Abstract*—In order to satisfy the high-performance and low-power requirements for advanced embedded systems with greater flexibility, it is necessary to develop chip multiprocessors. However, homogeneous multi-core systems do not achieve higher potential per watt when compared to heterogeneous architectures, as serialized code sections can be accelerated without programmer effort. Therefore, our work aims to determine the optimal number of cores of different types to provision the CMP with, such that the area and power budgets are met and the application performance is maximized. We consider general-purpose multi-threaded applications to evaluate single ISA heterogeneous multi-core architectures having two types of cores, 'big' and 'small', as a way to reduce processor power dissipation by switching cores to achieve target performance level. In order to efficiently assign threads to cores, a novel aggressive scheduling policy is proposed, which is based on statistical modeling of dynamic variations in instructions per cycle. We used Sniper multi-core simulator for performance modeling and McPAT for power modeling. For the 4, 8 and 16-core systems we modeled, using $1/4^{th}$ of the total number of cores as 'big' cores and rest as 'small' cores proved to be the most optimal design.**

I. Introduction

The increase in need for machines with higher performance, limited computational power and increase in complexity in the design of uniprocessor has been the driving force for increase in interest in design of multi-core architecture. A multi-core processor has multiple cores integrated on a single chip. They are mainly of two types, (i) a multi-core architecture where every core is just an image of the other, called homogeneous multicore, and (ii) when a set of cores may differ in area, performance, power dissipated etc, it is called heterogeneous multicore.

However, multi-core processor introduces a number of new challenges and metrics of interest. First, accurate models for the execution time of multi-threaded benchmark applications running on heterogeneous cores are not readily available, although these are essential for optimization. This is in contrast to the application specific-domain where the performance models are well defined, often using formal models of computation such as data-flow graphs. Second, typical multi-threaded applications, for example the applications from the SPLASH-2 [19] and PARSEC 2.1 [18] benchmark suites, can be executed with a variable degree of parallelism, i.e., a varying number of parallel threads, at run-time. The architectural synthesis algorithm must be aware of the optimal degree of parallelism and optimal run-time scheduling of threads to cores for each application. Also, from an empirical perspective, an important metric of interest is the performance benefit of heterogeneous versus homogeneous CMP architectures with increasing chip area.

The various design issues in multi-core architecture also include resource sharing, power consumption, performance, area of the cache, cache coherence etc. In order to harness the resources provided by a multicore architecture the application must show a certain level of parallelism. With additional cores, power consumption and heat dissipation become a concern and must be simulated before layout to determine the best floorplan which distributes heat across the chip, while being careful not to form any hot spots [8]. Distributed and shared caches on the chip must adhere to coherence protocols to make sure that when a core reads from memory it is reading the current piece of data and not a value that has been updated by a different core. This adds to the total power consumption as well.

To satisfy the high-performance and low-power requirements for advanced embedded systems with greater flexibility, it is necessary to develop parallel processing on chips by taking advantage of the advances being made in semiconductor integration [12]. Heterogeneous cores provide different power/performance tradeoff [3]. In order to benefit from heterogeneous multicore architectures, the scheduler needs to consider the power/performance asymmetry of heterogeneous multicore architectures when making a scheduling decision.

A program's hardware demands are governed by its inherent characteristics. For example, consider the instruction-dependency-distance distribution of a program (the number of instructions between the producer of a data and its consumer). As shown in Fig. 1 (a) reported by Chen et al [15], the SPEC benchmark *apsi* has a large percentage of instructions with long dependency distance, while *mcf* has a high percentage of instructions with short dependency distance [15]. These two opposite trends in dependency distance distribution indicate different amounts of instruction level parallelism (ILP) in these two programs, and hence different requirements of instruction issue width on the processor core. As shown in Fig. 1 (b), *apsi* demonstrates a near constant reduction rate in execution time as the instruction issue width is changed from 1 to 8. This is because the program has sufficient ILP, as indicated in the distribution, to keep up with the issue width scaling, and hence favors processor core with large issue width.

On the other hand, *mcf* has a significantly lower reduction rate in execution time, and the amplitude of the rate sharply goes down as the instruction issue width gets larger, which means the program is more suitable to run on a core with small issue width. Therefore, a program's inherent characteristics

determine its hardware resource demands, and this is used to find optimal number of cores of different types.

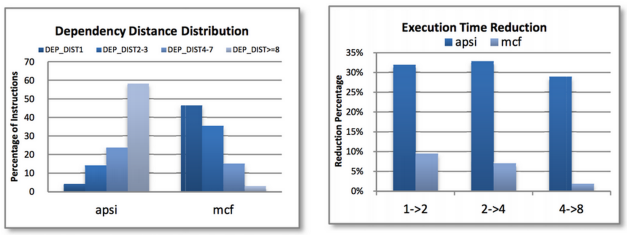


Fig. 1: Instruction dependency distance distribution and execution time reduction of *apsi* and *mcf*

The rest of the paper is organized as follows, Section II discusses about the related work, Section III describes the technical details, explaining why we need heterogeneous processors, our design and the scheduling policy we used. In Section IV we present our experimentation methodology, while in Section V we present results of our experimentation and its analysis. In Section VI we tell about our future work, milestones, and Section VII concludes the work.

II. Related Work

Considerable amount of work has been done on power-related optimizations for processor design. These can be broadly classified into two categories: (1) work that uses gating for power management, and (2) work that uses voltage and frequency scaling of the processor core to reduce power [2].

Gating-based power optimizations [20], [21] provide the option to turn off (gate) portions of the processor core that are not useful to a workload. However, for all these techniques, gating benefits are limited by the granularity of structures that can be gated, the inability to change the overall size and complexity of the processor. Chip-wide voltage and frequency scaling reduces the parameters of the entire core [22]. While this reduces power significantly, the power reductions are uniform across the portions of the core that are performance critical for this workload, and the portions of the core that are not. Furthermore, voltage and frequency scaling is fundamentally limited by the process technology in which the processor is built. Heterogeneous multi-core designs address both these deficiencies.

Fine-grained voltage/frequency scaling techniques using multiple clock domains have been proposed recently which obviate some of the disadvantages of conventional scaling-based techniques. However, similar to gating-based approaches, the benefits are likely to be limited by static leakage inefficiencies as well as the number of voltage domains that can be supported on a chip.

Core switching to reduce power was introduced previously in [21]. Recently, it has also been used for reducing power density in a homogeneous multiple-core architecture through the use of frequent core switches to idle processors.

General-purpose, heterogeneous CMPs were first proposed by Kumar et al. [1] in the context of single-threaded applications running on CMPs in a multi-programmed fashion. In [1], the authors propose hill-climbing based solutions for heterogeneous CMP design, but only for multi-programmed workloads. Gupta et al. [16] take uncore which is a collection of components of a processor not in the core but essential for core performance such as last level cache (LLC), integrated memory controllers (IMC), on-chip interconnect (OCI), power control logic (PWR), etc.. They take into account the dynamic thread mapping for heterogeneous multicore systems. Flicker [17], proposed by Petrica et al., takes a different approach by dynamically scaling core resources to create adaptive and configurable heterogeneity in hardware.

Turakhia et al. [14] propose 'HaDeS' in the domain of synthesis for heterogeneous multicore systems by formulating a non-linear optimization problem and developing a heuristic to transform the problem into integer linear program in polynomial time. Their heterogeneous architecture explains how it addresses issues by (i) efficiently exploring the entire design space of heterogeneous CMP architectures using a novel iterative optimization strategy, (ii) using a more realistic performance model that is validated against detailed simulations on both homogeneous and heterogeneous CMPs, and (iii) optimizing over a set of benchmark applications.

How to best schedule workloads (or threads) on the most appropriate core type is foundational to single-ISA heterogeneous multi-core processor. Making wrong scheduling decisions can lead to suboptimal performance and/or excess energy and power consumption. A significant body of work has focused on this scheduling problem for multi-program workloads. Some prior work uses workload memory intensity as an indicator to guide application scheduling [27], [28]. Others make offline scheduling decisions based on profiling information [28], [32], or use sampling-based solutions [27]. Model-based scheduling leverages analytical models to steer scheduling in order to overcome inaccuracies due to heuristics (e.g., memory dominance) and scalablity limits due to sampling [31]. Prior work on scheduling multi-threaded workloads on heterogeneous multicores has primarily focused on accelerating the serial fraction of code [26] and critical sections [33], and on identifying and accelerating critical bottlenecks [30] and threads using synchronization behavior [29].

All of the prior work in scheduling heterogeneous multicores focused on optimizing total system throughput. PIE [34] is a recently proposed scheduling framework to predict workload-core mappings that improve performance. It incorporates hardware parameters and CPI stack to predict the ratio of memory level parallelism and the ratio of instruction level parallelism between the currently running core and another core. It scales to more than two types of cores. However, the framework doesnt explicitly model power consumption and thereby is not fully power-aware. It does not provide an efficient solution for obtaining optimal mapping for largescale heterogeneous systems either.

Overall, having heterogeneous processor cores provides potentially greater power savings compared to previous approaches and greater flexibility and scalability of architecture design. As of now, there has not been much work on finding optimal number of cores of different types which this paper tries to solve such that performance needs as well as energy constraints are satisfied within the user area budget.

III. Technical Details

To attain high-performance and low-power requirements for advanced embedded systems it is necessary to develop parallel processing on chips by taking advantage of the advances being made in semiconductor integration. The subsequent sections explain the reason for choosing heterogeneous processors, proposes a design to get optimum value for power/performance and then describes the scheduling policy which is used.

A. Why Heterogenous?

In an ideal homogeneous architecture, processing capability and power consumption are evenly distributed among its cores. Thus the decrease of computation time and the increase of system power are linear with the number of active cores. So the total energy efficiency (the Energy-Delay^2 product) is constant. However, these assumptions are not valid. First, as per Amdahl's law, performance improvement suffers from the law of diminishing return such that speedup may not scale linearly with the number of cores. Second, not all energy consumed by the system contributes to useful computation. Power consumption overhead is not linear with the number of active cores. Third, the achievable performance is limited by architectural factors such as I/O bandwidth and cache size.

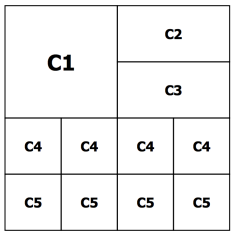


Fig. 2: Example Heterogeneous Design

In a heterogeneous multicore architecture a core may differ from the other cores in many ways. First, we focus on the performance aspect of multicores. Fig. 2 shows an example of a heterogeneous multi-core architecture on a single chip. The parallelizable portion of the code can be executed on smaller cores C2, C4, C5, whereas large critical sections can be executed on the large core C1. Building a performance oriented heterogeneous core is desired because many simple cores together provide high parallel performance while complex cores help in providing high serial performance. Amdahl's Law is still relevant as we enter a heterogeneous multi-core computing era [9]. Amdahl's Law is a simple analytical model that helps developers to evaluate the actual speedup that can be achieved using a parallel program. If p is the number of processors and α is the parallelizable fraction of a program, then Amdahl's law states that speedup is given by

$$Speedup = \frac{1}{\frac{\alpha}{p} + (1 - \alpha)}$$

B. Design

A single processor will contain many small simple cores, and a few larger complex cores. Simple cores require less area as they will be scalar, in-order and might have a smaller cache and lower clock frequency. Complex cores will have superscalar design, higher issue width, larger ROB size and may be equipped with high-performance features such as out-of-order instruction scheduling, aggressive prefetching or a vector unit. However, they will be larger and will consume significantly more power. Heterogeneous architectures are motivated by their potential to achieve a higher performance per watt as compared to homogeneous systems because each application can run on a core that best suits its state. For example, a gaming app can run on a simple core when user is setting configurations or have paused a game, but the application should switch to complex core once the actual intensive gaming starts.

Our work primarily focuses upon optimizing power and performance with respect to area and number of cores of different types. A new performance model is designed for multi-threaded applications from the SPLASH-2 [19] and PARSEC-2.1 [18] benchmark suites. The performance model for applications is scheduled on both homogeneous and heterogeneous CMPs, and across a wide range of chip sizes and number of chips. We calculate the optimal number of cores of each type based on evaluation of multiple libraries of configuration within the area budget. We then feed the performance modeling parameters to McPAT which calculates the energy and power efficiency.

In a heterogeneous CMP, the individual execution time of each parallel thread is different. In the parallel phase of execution, threads typically synchronize on a barrier, i.e., all threads must finish execution before the application can proceed to the next phase. Therefore, the latency of a parallel phase is determined by the worst case execution latency across all parallel threads.

C. Scheduling

To take full advantage of heterogeneous CMPs, the system software must use the execution characteristics of each application to predict its future processing needs and then schedule it to a core that matches those needs if one is available [1], [2]. The predictions can minimize the performance loss to the system as a whole rather than that of a single application. Recent work has shown that effective schedulers [6] for heterogeneous architectures can be implemented and integrated with current commercial operating systems.

Let's assume the hardware consists of B big cores and S small cores. The adaptive scheduler we propose selects a thread running on big core with lowest IPC, *thread_big*, and selects a thread running on small core with highest IPC, *thread_small*. If the IPC of *thread_big* is lesser than the dynamic mean IPC (as shown in Eq. 1) of the big core, the thread is assigned to the small core and in case IPC of *thread_small* is greater than the current core's sum of mean and variance, Eq. 1 + Eq. 2, it is moved to the big core. Each thread is pinned to a specific core and cores are handed out to new threads in round-robin fashion. If multiple threads share a core, they are time-shared with a configurable quantum.

For the dynamic thresholding as mentioned above, online incremental algorithm is used [25]. This reduces overhead of calculations as new value depends on the previous computed

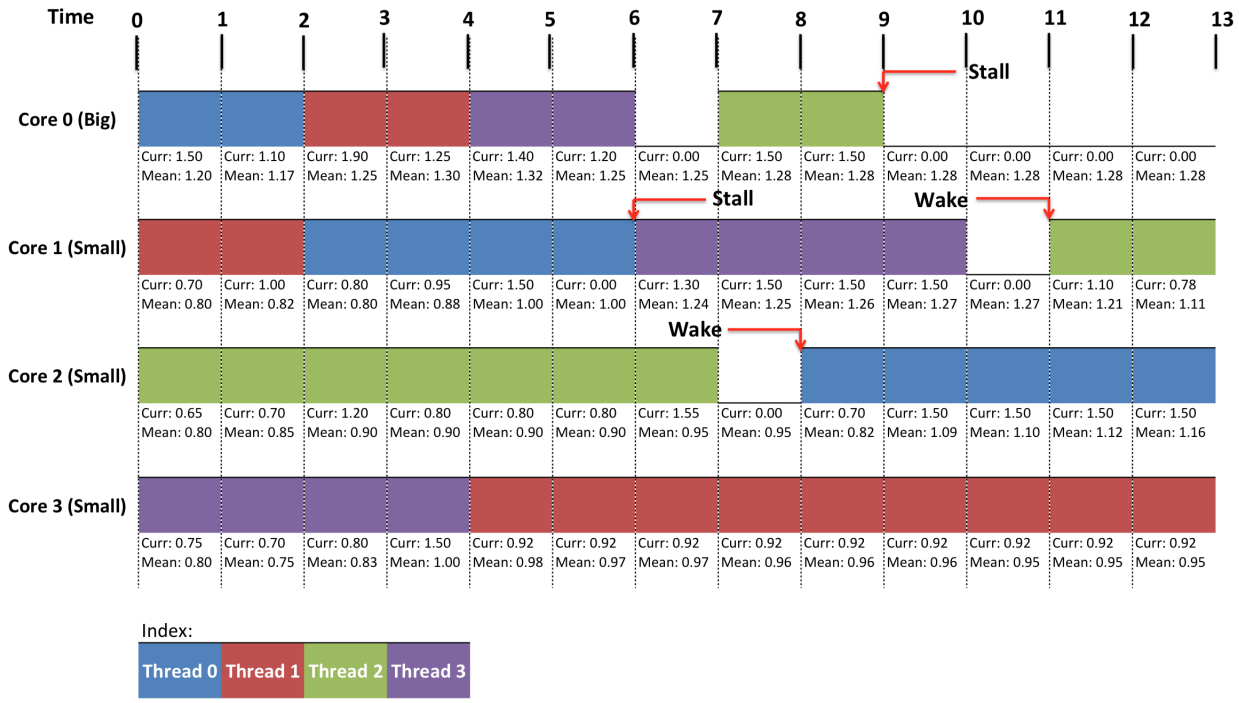


Fig. 3: Dynamic behavior of IPC based scheduler

value. Using this algorithm, the mean of instructions per cycle is calculated as,

$$\overline{ipc_n} = \overline{ipc_{n-1}} + \frac{ipc_n - \overline{ipc_{n-1}}}{n} \quad (1)$$

and variance as,

$$\sigma_n{}^2 = \frac{(n-1)\sigma_{n-1}^2 + (ipc_n - \overline{ipc_{n-1}})(ipc_n - \overline{ipc_n})}{n} \quad (2)$$

```
for core in NUM_CORES
    if (core is BIG) and (thread_core != INVALID)
        if (core_ipc_i < big_ipc)
            big_ipc = core_ipc
    if (core is SMALL) and (thread_core != INVALID)
        if (core_ipc_i > small_ipc)
            small_ipc = core_ipc

BIG_THRESHOLD = MEAN(big_core_ipc)
SMALL_THRESHOLD = MEAN(small_core_ipc) + VARIANCE(small_core_ipc)

if (big_ipc < BIG_THRESHOLD) and (small_ipc > SMALL_THRESHOLD)
    moveToSmall(thread_big)
    moveToBig(thread_small)
else if (big_ipc < BIG_THRESHOLD) and (small_ipc < SMALL_THRESHOLD)
    moveToSmall(thread_big)
```

Fig. 4: *Pseudo Code*

Since we are interested in the highest performance that a processor can offer, we assume static mapping of n threads on n different cores. Thus, the performance of n particular threads is the performance of the best static mapping. However, this actually represents, in some sense, a lower bound on performance.

Fig. 3 shows a simple window of the scheduling algorithm for a 4-core system with 1 'big' core and 3 'small' cores in action. The currently running thread on each core at any particular time instant is shown in the timing diagram. The scheduling algorithm calculates the current IPC at a particular instant (shown as *Curr*) and the present value of *Mean* is also shown. Rescheduling to the big core is done when the current IPC of the running thread is greater than the sum of mean and variance at that instant. In the case when there are multiple threads eligible to shift to the big core, the one with higher IPC is preferred. As shown, no context switch happens up to $t = 1$ since the current IPC of any of the cores did not increase beyond the threshold (mean + variance).

After $t = 2$, thread 1 context switches to big core 0 as its IPC increases beyond the *SMALL_THRESHOLD*, which is mean of IPC as shown in pseudo code in Fig. 4. Thread 0 is switched to the small core 1 due to the drop in its IPC. After $t = 4$, the hike in thread 3's IPC and drop in thread 1's IPC leads to it getting context switched to the big core. At $t = 4$, even though thread 0 has a higher IPC than the *SMALL_THRESHOLD*, context switch is avoided since thread 3 has a high IPC beyond the threshold ultimately avoiding unnecessary context switches. At $t = 6$ thread 0 stalls and thread 3 has a drop in its IPC leading to the context switch to the smaller core. Now the big core is running in power saving idle mode.

At $t = 7$, thread 2 context switches to the big core and small core 2 runs in idle mode. Later when thread 0 wakes up, it runs on the small core by default, assuming that the I/O bound process does not run a high IPC task after it is awoken from a stall compared to a processor bound task. Similarly at $t = 11$, thread 2 wakes up from a memory stall on small core

TABLE I: Energy-Delay-Area values for various configurations

Num Cores	Config (big_small)	Energy x Delay^2				Area (mm^2)	Normalized Average Energy x Delay^2 x Area
		blackscholes (x 10^{14})	fuildaniumate (x 10^{17})	fft (x 10^{13})	radix (x 10^{13})		
16	15_1	1.1696	598.7809	1.8014	6.1910	761.554	35.6723
	14_2	2.0773	92.7541	1.2255	2.9058	757.104	6.2012
	13_3	1.4269	38.3247	1.3798	2.3575	752.655	2.9196
	12_4	2.5741	19.2564	1.3652	2.3160	748.206	2.0224
	11_5	2.8727	10.7049	1.3318	2.5155	743.756	1.5959
	10_6	2.1110	7.5870	2.2336	2.2412	739.307	1.4081
	9_7	2.3234	5.5930	3.5010	2.3347	734.857	1.5562
	8_8	2.3814	4.0203	3.2315	2.2391	730.408	1.4142
	7_9	2.4476	4.0118	2.4015	2.4001	725.958	1.2890
	6_10	2.4501	4.1194	1.2706	2.2257	721.509	1.0773
	5_11	2.4420	4.1677	1.3857	2.2040	717.059	1.0895
	4_12	**2.2038**	**3.7206**	**1.3035**	**2.1631**	**712.609**	**1.0000**
	3_13	2.5643	4.0579	1.2107	2.3439	708.159	1.0744
	2_14	2.3781	4.1077	1.1867	2.2965	703.709	1.0304
	1_15	2.1352	4.2442	1.3490	2.2131	699.26	1.0096
8	7_1	4.6949	438.0272	5.7814	83.0542	379.625	10.2701
	6_2	13.5515	66.2091	6.2983	28.9947	375.169	2.7224
	5_3	10.0780	22.4977	7.3524	16.1237	370.713	1.4996
	4_4	11.5269	10.8956	4.8122	14.5415	366.257	1.2090
	3_5	11.8579	10.9323	5.3272	10.8768	361.801	1.1194
	2_6	**8.9977**	**10.8445**	**8.0477**	**7.4979**	**357.345**	**1.0000**
	1_7	9.3284	10.8841	9.8526	5.9763	352.889	1.0204
4	3_1	49.8095	264.6146	35.1995	449.5838	187.079	6.7433
	2_2	45.0964	40.6926	26.2432	77.6371	182.629	1.6571
	1_3	**40.2037**	**40.4143**	**16.1735**	**28.5866**	**178.179**	**1.0000**

1. Since no task has high IPC, the big core continues in power saving idle mode.

IV. Experiment Setup

We used Sniper [10] for performance modeling. It is a modular platform for computer system architecture research, encompassing system-level architecture as well as processor microarchitecture. It also has easier interface for multiple heterogeneous core modeling. We initially wanted to use gem5 [4] for performance modeling, because of its precision, but it currently does not provide much flexibility for heterogeneous core modeling.

McPAT [11] was used for power modeling as it has integrated power, area, and timing modeling framework for multicore architectures. All analysis is done for 45nm technology node and nominal 1.2V supply voltage.

The architectural design was tested using multi-threaded PARSEC-2.1 and SPLASH-2 [18], [19] benchmark suites. These benchmark suites focus on emerging workloads and has been designed to be representative of next-generation shared-memory programs for chip-multiprocessors.

Currently, we used two types of cores, i.e. 'big' and 'small'. Table II describes the configuration we used for each of them. We also consider multi-threaded workloads and assume that each core runs at most one thread at a time - in other words, we experiment for single threaded execution, without simultaneous multithreading (SMT). Without loss of generality, the total number of threads is assumed to be n, identical to the total number of cores.

V. Experimental Results

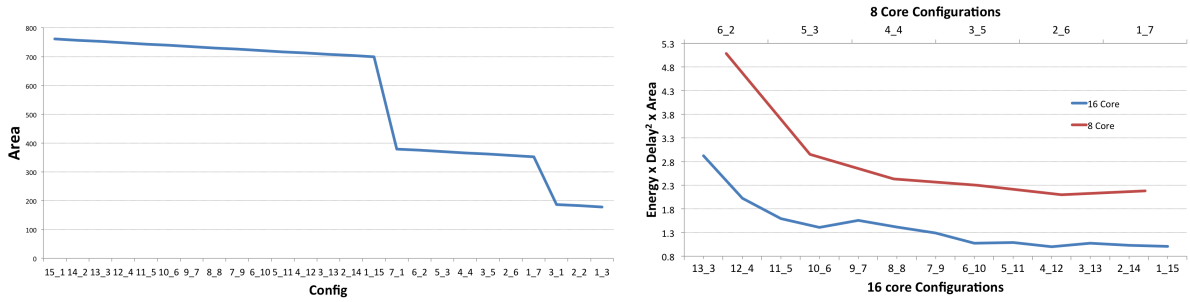
We modeled all possible heterogeneous combinations for 4, 8 and 16-core system and calculated the time-delay and energy consumption for each benchmark. The results are tabulated in Table I. Please note that processor configuration has been denoted as {num_bigs}_{num_smalls}, for example, a system with 3 big cores and 1 small core is referenced as 3_1 in the rest of the paper. The time delay reported is the maximum of all the processors as latency of a parallel phase is determined by the worst case execution latency across all parallel threads. Normalized Energy x Delay^2 x Area is also reported and plotted in Fig. 5b to better analyze performance - energy trade-off within the user area budget. The most optimal design would reduce Energy x Delay^2 x Area.

TABLE II: Configuration

Core Parameter	Big	Small
Dispatch width	4	2
L1-I/D Cache size	64 KB	32 KB
L2 Cache (private)	256 KB	128 KB
Frequency	2.66 GHz	1.50 GHz
ROB Window	128	32

Analysis

As expected, we see larger Energy x Delay^2 for systems with more number of 'big' cores. Detailed results are shown in Table I. Users with a known area budget can find maximum number of cores that could be used from Fig. 5a . Once, maximum number of cores possible is known, Fig. 5b helps to know the most optimal configuration, i.e. how many 'big' and 'small' cores should be used. If the user requires additional performance benefit, even at the cost of excess energy consumption, Fig. 5b could help find the most optimal configuration for their particular application. The dynamic behavior of scheduler helps it to adapt to any benchmark being run.



(a) Area v/s Configuration

(b) (Energy x Delay^2 x Area) for 8 and 16 core configurations

Fig. 5: Finding best configuration for the user, (a) helps to know maximum number of cores possible within the user area budget, and (b) helps to find optimal configuration for specified number of cores

From Fig. 5b shows that configuration 4_12, for 16-core system, works out to be optimal configuration in terms of Energy x Delay^2 x Area product. Similarly, 2_6 and 1_3 configuration give the most optimal performance and energy trade-off.

VI. Future Work

In the near future additional support for larger configuration library of cores can be given, i.e. also support multiple 'medium' size cores to achieve even better energy-delay trade-off. Also, the work was done entirely in Snipersim, which currently does not support simultaneous multi-threading (SMT), thus it lacks the ability to represent the modern architectures. In addition to that, Snipersim is not as detailed in its simulation and functionalities than other available simulators like gem5. So in order to achieve more precise results, using tools like gem5 would be beneficial, though it does have a lot of hurdles associated with it.

Table III provides the milestone we followed during the span of 18-743 course and we were able to complete our 'Sky' goal.

TABLE III: Milestone

Stage	Status/ Date	Progress
Initial	Done	Setup Snipersim and McPAT
Midway	Done	Complete heterogeneous design
75%	Done	Measure power and performance for single heterogeneous configuration
Final	Done	Obtain optimal power/performance sweet-spot for user specified area and energy budget
Sky	Done	Aggressive scheduling policy

VII. Conclusion

We were able to successfully create a power-performance model for heterogeneous multicore systems and predict the optimal configuration for a particular area budget. The proposed scheduler was able to optimize Energy x Delay^2 x Area product based on dynamic IPC variations in the benchmark. Statistical modeling of IPC makes it adaptive to any user application. Based on user's area budget, an appropriate configuration can be chosen using this model. Having $1/4^{th}$ of the total cores as large sized cores proved to maintain the balance between the power-performance tradeoff. A user can give the area as an input to this model and use it obtain the configuration and size of cores required. This can be customized to focus on energy or performance and obtain a suitable chip core configuration.

VIII. Work Distribution

Naman: Setup Sniper, making modifications to the simulator, python scripting to run multiple simulations, writing report.
Prashanth: Analyzing related work, scripting to capture performance and power, plotting graphs, writing report.

References

[1] Kumar, Rakesh, Dean M. Tullsen, and Norman P. Jouppi. "Core architecture optimization for heterogeneous chip multiprocessors." Proceedings of the 15th international conference on Parallel architectures and compilation techniques. ACM, 2006.

[2] Kumar, Rakesh, et al. "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction." Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. IEEE, 2003.

[3] Lukefahr, Andrew, et al. "Composite cores: Pushing heterogeneity into a core." Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2012.

[4] Binkert, Nathan, et al. "The gem5 simulator." ACM SIGARCH Computer Architecture News 39.2 (2011): 1-7.

[5] Mutlu, Onur, "Asymmetry" 18-740 Lecture 2.1 Fall 2013

[6] Ghiasi, Soraya, Tom Keller, and Freeman Rawson. "Scheduling for heterogeneous processors in server systems." Proceedings of the 2nd conference on Computing frontiers. ACM, 2005.

[7] Tsoi, Kuen Hung, and Wayne Luk. "Power profiling and optimization for heterogeneous multi-core systems." ACM SIGARCH Computer Architecture News 39.4 (2011): 8-13.

[8] Hyari, Abeer. "A comparative study on heterogeneous and homogeneous multiprocessors." University of Jordan (2009).

[9] Marowka, Ami. "Extending Amdahl's Law for Heterogeneous Computing." Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on. IEEE, 2012.

[10] Carlson, Trevor E., Wim Heirman, and Lieven Eeckhout. "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation." Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011.

[11] Li, Sheng, et al. "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures." Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on. IEEE, 2009.

[12] Uchiyama, Kunio, et al. Heterogeneous Multicore Processor Technologies for Embedded Systems. Springer, 2012.

[13] Liu, Guangshuo, Jinpyo Park, and Diana Marculescu. "Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems."Computer Design (ICCD), 2013 IEEE 31st International Conference on. IEEE, 2013.

[14] Turakhia, Yatish, et al. "HaDeS: architectural synthesis for heterogeneous dark silicon chip multi-processors." Proceedings of the 50th Annual Design Automation Conference. ACM, 2013.

[15] Chen, Jian, and Lizy Kurian John. "Energy-aware application scheduling on a heterogeneous multi-core system." Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on. IEEE, 2008.

[16] V. Gupta, P. Brett, D. Koufaty, and D. Reddy, The forgotten uncore: On the energy-efficiency of heterogeneous cores, in Proc. USENIX Annual Technical Conf, 2012.

[17] P. Petrica, A. Izraelevitz, D. Albonesi, and C. Shoemaker, Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems, in Proc. ISCA, 2013.

[18] Bienia, Christian, et al. "The PARSEC benchmark suite: Characterization and architectural implications." Proceedings of the 17th international conference on Parallel architectures and compilation techniques. ACM, 2008.

[19] Woo, Steven Cameron, et al. "The SPLASH-2 programs: Characterization and methodological considerations." ACM SIGARCH Computer Architecture News. Vol. 23. No. 2. ACM, 1995.

[20] Folegnani, Daniele, and Antonio Gonzlez. "Reducing power consumption of the issue logic." In Workshop on Complexity-Effective Design. 2000.

[21] Ghiasi, Soraya et al. Using IPC Variation in Workloads with Externally Specified Rates to Reduce Power Consumption. 2000.

[22] Govil, Kinshuk, Edwin Chan, and Hal Wasserman. "Comparing algorithm for dynamic speed-setting of a low-power CPU." Proceedings of the 1st annual international conference on Mobile computing and networking. ACM, 1995.

[23] Heo, Seongmoo, Kenneth Barr, and Krste Asanovic. "Reducing power density through activity migration." Low Power Electronics and Design, 2003. ISLPED'03. Proceedings of the 2003 International Symposium on. IEEE, 2003.

[24] Spiliopoulos, Vasileios, et al. "Introducing DVFS-management in a full-system simulator." Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 2013.

[25] Donald E. Knuth. The Art of Computer Programming, volume 2: Seminumerical Algorithms, 3rd edn., p. 232. Boston: Addison-Wesley, 1998.

[26] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahls law through EPI throttling. In Proceedings of the International Symposium on Computer Architecture (ISCA), pages 298309, June 2005.

[27] M. Becchi and P. Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. Journal of InstructionLevel Parallelism (JILP), 10:126, June 2008.

[28] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. In Proceedings of the 46th Design Automation Conference (DAC), pages 927930, July 2009.

[29] K. Du Bois, S. Eyerman, J. B. Sartor, and L. Eeckhout. Criticality stacks: Identifying critical threads in parallel programs using synchronization behavior. In Proceedings of the International Symposium on Computer Architecture (ISCA), pages 511522, June 2013.

[30] J. Joao, M. Suleman, O. Mutlu, and Y. Patt. Bottleneck identification and scheduling in multithreaded applications. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 223234, Mar. 2012.

[31] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke. Composite cores: Pushing heterogeneity into a core. In Proceedings of the International Symposium on Microarchitecture (MICRO), pages 317328, Dec. 2012.

[32] D. Shelepov, J. C. S. Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar. HASS: A scheduler for heterogeneous multicore systems. Operating Systems Review, 43:6675, Apr. 2009.

[33] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. Accelerating critical section execution with asymmetric multicore architectures. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 253264, Mar. 2009.

[34] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, Scheduling heterogeneous multi-cores through performance impact estimation (PIE), in Proc. ISCA, 2012.