Spendthrift: Machine Learning Based Resource and Frequency Scaling for Ambient Energy Harvesting Nonvolatile Processors

Kaisheng Ma[1], Xueqing Li[1], Srivatsa Rangachar Srinivasa[1], Yongpan Liu[2], John Sampson[1], Yuan Xie[3], and Vijaykrishnan Narayanan[1]
{kxm505, lixueq, sxr5403, sampson, vijay}@cse.psu.edu, ypliu@tsinghua.edu.cn, yuanxie@ece.ucsb.edu
[1]*Dept. of Computer Science and Engineering, Pennsylvania State University*
[2]*Dept. of Electronic Engineering, Tsinghua University*
[3]*Dept. of Electrical and Computer Engineering, University of California at Santa Barbara*

Abstract - Batteryless energy harvesting systems face a twofold challenge in converting incoming energy into forward progress. Not only must such systems contend with inherently weak and fluctuating power sources, but they have very limited temporal windows for capitalizing on transitory periods of above-average power. To maximize forward progress, such systems should aggressively consume energy when it is available, rather than optimizing for peak average-case efficiency. However, there are multiple ways that a processor can trade between consumption and performance. In this paper, we examine two approaches, frequency scaling and resource scaling, and develop a predictor-driven scheme for dynamically allocating future power budgets between the two techniques. We show that our solution can achieve forward progress equal to 2.08X of the baseline Out-of-Order (OoO) processor with the best static configuration of frequency and resources. The combined technique outperforms either technique in isolation, with frequency-only and resource-only approaches achieving 1.43X and 1.61X forward progress improvements, respectively.

Keywords: Nonvolatile processor; energy harvesting; machine learning; power-adaptive microarchitecture; Internet of Things.

1. INTRODUCTION

With the development of nonvolatile processors (NVPs), energy harvesting is emerging as an increasingly attractive means for powering the internet of things (IoT) [1,2,4]. NVPs can handle unstable input power by backing up the computation state in distributed nonvolatile flip-flops or integrated memories [1] at very short timescales, allowing systems using these processors to operate without large energy storage devices.

In energy-harvesting systems, the local variance in input power is large: The peak available power can be many times larger than average power, and there can be sustained periods where only a minimally active processor can operate at all. Incorporating flexibility into a processor to adapt to changing conditions is a long-studied area. Techniques such as Turbo Boost [8,9] and other dynamic voltage and frequency scaling [10-13] as well as microarchitectural resource adaptation techniques [14-15] have been proposed by prior work in the context of energy-efficient computing. Prior work on energy harvesting NVPs [1-7] has also indicated that no single microarchitecture best translates input energy into forward progress across varying input power traces. Conceptually, the ideal NVP design is the one that can operate in input power valleys for more on-duty time, and also convert input energy plateaus into more progress rather than let them leak away or overflow

Both frequency-scaling and microarchitectural adaptation are promising approaches for consuming energy that cannot be otherwise stored in a batteryless system. However, which approach is preferable and how the two approaches can synergize have not been explored in the context of NVPs for the IoT space. In particular, the policy space can be seen as a combination of predicting a) energy income in the next epoch and b) among designs capable of consuming as much of the energy income as possible over the coming epoch, which will offer the best forward progress per unit energy.

The aims of this paper are to explore the effectiveness of both frequency and resource scaling techniques in the context of NVPs, and to develop an effective dynamic prediction mechanism to set both frequency and resource parameters efficiently. Our work makes the following contributions:

- Targeting lower energy per instruction (EPI) for NVP, we propose using an infrequently executed (5Hz) neural network-based predictor to manage bottleneck resources in a reconfigurable out-of-order processor.
- Targeting aggressive leveraging of harvested energy for forward progress, we design a machine learning based dynamic frequency scaling (DFS) module for nonvolatile processors.

The rest of the paper is organized as follows. Section 2 proposes a NVP bottleneck resource predictor and Section 3 presents the smart NVP frequency scaling architecture. Section 4 describes simulation infrastructure and methodology. Section 5 presents the simulation results and analyzes the benefits of combining resource and frequency scaling predictors. We discuss the prior work in the field in Section 6 and conclude with Section 7.

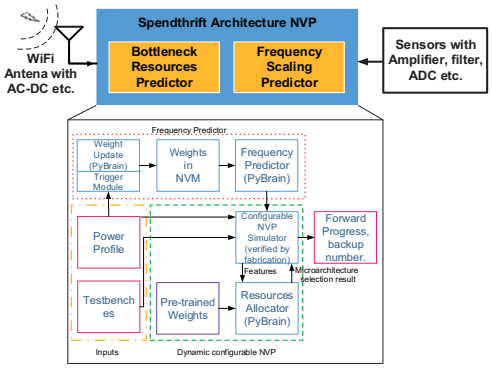

Figure 1. Spendthrift architecture and simulation structure.

2. SPENDTHRIFT: BOTTLENECK RESOURCE PREDICTIONS

In this section, we introduce the proposed hybrid processor microarchitecture, Spendthrift, which incorporates a single-issue In-Order microarchitecture and a multi-issue high performance OoO microarchitecture. The In-Order microarchitecture can

*This work was supported in part by NSF awards 1160483 (ASSIST), Center for Low Energy Systems Technology (LEAST) sponsored by MARCO and DARPA, NSFC Grant 61674094, Beijing Innovation Center for Future Chip, NSF 1500848, 1533933, a grant from Qualcomm, and U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award number DE-SC0013553.

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

operate with the minimum resources powered on for the minimum start-up threshold; The OoO microarchitecture operates with more aggressive power consumption but can achieve the highest throughput. The mechanism of how to find the best configuration for the maximum forward progress is also described in this section.

2.1 Resource Allocation System Structure

Figure 2 shows the system diagram. We select 10 adjustable potential bottleneck resources to balance EPI and performance. The total number of all possible configuration entries is 1024, and each entry uses 10 configuration bits. With limited power income in energy harvesting systems, we power on only bottleneck resources so as to boost performance with the minimum power penalty. With more resources powered on, the instruction per clock-cycle (IPC) may increase a lot, but the power consumption does not increase as much, thus EPI reduces, as shown in Figure 3 and Figure 4.

It is also noted that turning on and off resources results in switching delay and energy for power-gating control [27]. In addition, in our proposed solution, we have considered the need of freeing the resources before turning them off.

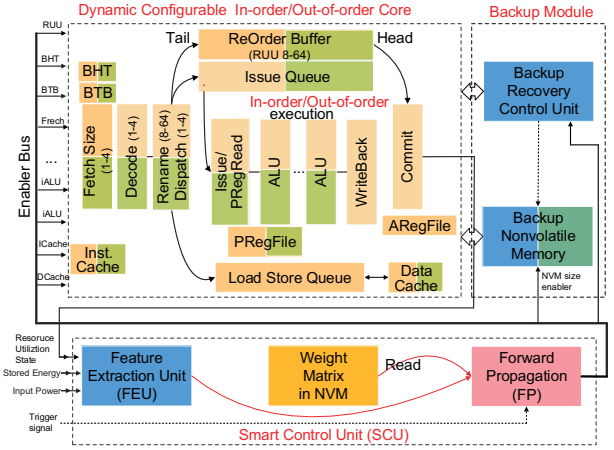

Figure 2. The system diagram of configurable resource allocation architecture. Configurations: IO/OO: InOrder/Out-of-Order, low for In-Order, high for Out-of-Order; FT: Fetch Width, low for 1, high for 4; DC/IS: decoder and issue width, low for 1, high for 4; RUU: low for 8, high for 128; ALU: low for 1, high for 4; MP: memory port, low for 2, high for 8; CL1: Instruction and Data Cache: low for -cache:il1 il1:256:32:1:l, high for -cache:il1 il1:256:32:4:l; ICL2: low for -cache:dl1 dl1:256:32:1:l, high for -cache:dl1 dl1:256:32:4:l; DCL2: low for -cache:dl2 ul2:64:64:4:l, high for -cache:dl2 ul2:256:64:4:l; PRE: low for -bpred:bimod 128, high for -bpred:bimod 1024.

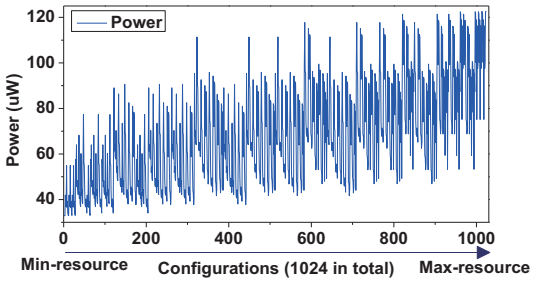

Figure 3. NVP power consumption for different resource configurations, testbench "susan_corners".

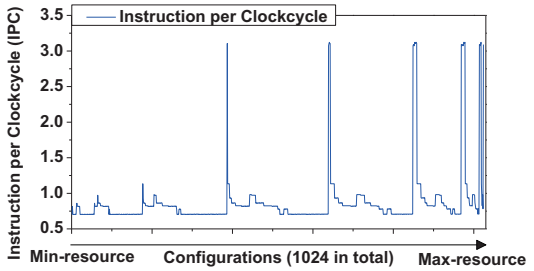

Figure 4. IPC V.S. different resource configurations, testbench "susan_corners".

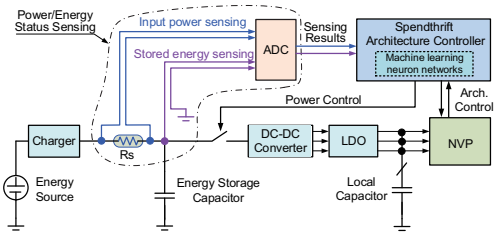

Figure 5. System diagram with feature extraction circuits.

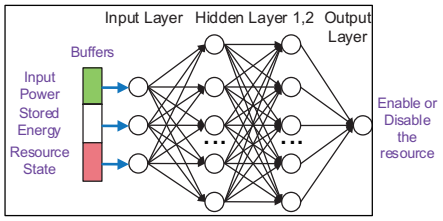

Figure 6. Neural network for one resource prediction.

2.2 Feature Extraction and Neural Networks

Rather than building a single, large neural network that predicts 10 resources at a time, spendthrift uses 10 small neural networks, one network for one resource prediction. The reason is that multiple small neural networks can significantly reduce the computation amount. Each neural network has four layers as shown in Figure 6: one input layer, two hidden layers, and one output layer. These are optimized results following a similar approach considering number of layers and numbers of neurons in each layer, as discussed by Ma. et al. [3].

In the input layer, there are three inputs. One input is for the current resource usage conditions: "0" indicates a not fully utilized resource, and "1" indicates a fully utilized resource. A condition of "1" indicates one possible bottleneck resource as it may need extra resources for further performance improvement. The other two inputs are the input power and the stored energy. Both are captured through the front-end circuits shown in Figure 5 every 0.2 second by an A/D converter (ADC). A small resistor Rs is used to sense the power delivered through it with negligible voltage drop. Considering 32 required sensed levels, a 5-bit resolution is sufficient while consuming only 1nW power. As for the stored energy sensing, it is equivalent to a measurement and calculation of the voltage across the energy storage capacitor.

The two hidden layers each consists of 10 hidden neurons. The output has 1 node for resource selection result. When the output is higher than 0.5, that resource is treated as a bottleneck resource and will be enabled. The neural networks are triggered every 0.2s. Offline training is used with 10k training set, achieving an accuracy above 90% on Mibench "small inputs" [16] and these initial trained weights are stored in the NVM.

3. DYNAMIC FREQUENCY SCALING

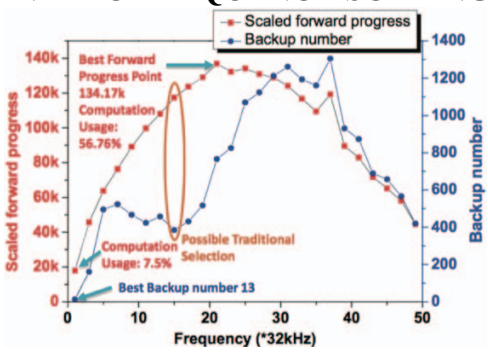

Figure 7. NVP performance vs. frequency with minimum resources.

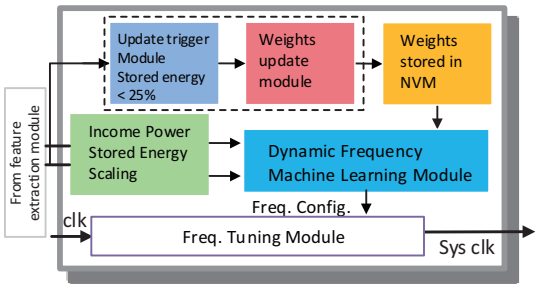

Figure 8. Proposed dynamic frequency scaling structure.

In this section, we investigate NVP frequency scaling policies as another effective approach. For simplicity without losing generality, we use a fixed number of resources, and three power profiles, a typical example of which is seen in Figure 13.

3.1 Performance vs. Static Frequency

We scan across frequencies from 32kHz to 1.5MHz with a step of 32kHz. In the simulations, a 672kHz static frequency results in the best forward progress, as shown in Figure 7. Compared to the forward progress with minimized 32kHz frequency, the forward progress is accompanied by the penalty of 29X more backup operations.

With too low a frequency, a large portion of harvested energy leaks away or cannot be stored in the capacitor because of power consumption lower than input power. With too high a frequency, more power overhead is consumed because of more backup and recovery penalties. Both lead to reduced forward progress. In subsequent sections, the best static frequency 672kHz is used as the comparison baseline for dynamic frequency scaling solution.

3.2 Proposed DFS Architecture

Figure 8 shows the diagram of the proposed DFS architecture, as a part of the controller integrated in the system clock path. The dynamic frequency machine learning module generates the frequency configuration signal for frequency tuning module. The machine learning module consists of a forward propagation network with scaled income power and stored energy as inputs. The initial weights are trained offline and stored in NVM.

As the processor power consumption varies greatly with the amount of resources being used, as shown in Figure 3, the proposed DFS architecture adapts to the resources allocation policies. Once the stored energy level is less than 25% of full stored energy while the income power is still able to power a 32kHz processor with minimum resources, the weights update module is triggered to update the weights with a one-step lower frequency than the current one.

3.3 DFS Neural Networks

This neural network decides the best frequency that the NVP with fixed resources should run at, based on the input power and the stored energy. The neural network is similar to the structure in Figure 6. It has 2 entries: power income level and stored energy level. There are two hidden layers for 32*2 hidden neurons, and 32 outputs as the predicted possibility to select the frequency. The frequency with the largest possibility will then be selected (Simplified Softmax layer). The initial neural networks achieve above 98% accuracy with <10k training set.

4. METHODOLOGY

4.1 Simulation Infrastructure

The simulation infrastructure consists of several parts as shown in Figure 1: (i), the inputs of power profiles and testbenches; (ii), a bottleneck resource predictor implemented based on Pybrain [26]; (iii), a frequency predictor for NVP. (4), dynamic configurable NVP. The NVP simulator was proposed by Ma. et al.[1], and has been verified by a fabricated NVP [1,4].

The NVP simulator provides features like resource usage, power and energy level, to the resource allocator. By combining these features and pre-trained weights, the allocator gives feedbacks to NVP with microarchitecture selection results. The NVP uses these configurations to reduce energy per instruction and maximize the forward progress. For each given power profile and testbench, the simulation results are forward progress, etc. The frequency predictor generates frequency configuration predicted results for maximum energy used for computation. In order to integrate with the bottleneck resource predictor, the frequency predictor also has an online weights update module to adapt itself to the unstable microarchitecture.

4.2 Testbenches and Power Profiles

We use MiBench [16] on “large inputs” as our core evaluation suite. In addition to Mibench, we also use some neural network algorithms as testbenches [17]: ADALINE: Adaline network for pattern recognition, classification of digits 0-9 [18]. ART1: Adaptive resonance theory network, brain modeling stability-plasticity demonstration [19]. BAM: Bidirectional associative memory, heteroassociative memory association of names and phone numbers [20]. BOLTZMAN: Boltzmann machine [21]. BPN: Back-propagation, time-series forecasting [22]. CPN: Counter-propagation network, determination of the angle of rotation [23]. HOPFIELD: Hopfield model, associative recall of images [24]. SOM: Self-organizing map, reinforcement learning approach [25].

The power profiles are WiFi home/office profiles [1], measured in real home/office environments.

4.3 Overhead Analysis

Making predictions and effecting the changes in resource configurations and frequency imposes some overheads. We use one neural network prediction module to predict bottleneck resources one by one, and then the frequency prediction. We implement the neural network predictor using dedicated hardware, as the software overhead would be untenable. We evaluate the overhead of the predictor by synthesizing the prediction module using a 32nm library with VDD=0.85V. The neural network serial architecture shown in Figure 9 has only one multiply accumulate

(MAC) module, and a state machine is developed to select the weights, source neurons, and target neurons from the ROM or register files. The neural network predictor can run at a maximum of 156MHz, but we run it at 10MHz, considering the low frequency of the NVP. The power is 3.36uW, and it costs 141 cycles to finish one prediction for one bottleneck resource prediction. The energy cost of making one resource prediction is 47.36pJ. Similarly, the frequency predictor costs 711pJ per prediction. These overheads in energy correspond to 5.9% of average WiFi energy income during 0.2s interval (0.2s*10uW=2uJ). The additional power and energy sampling circuits also impose overheads, but we consider these negligible due to only being employed once per 0.2s.

The area is 23744um^2, 2.3% of a Non-pipelined processor. A single prediction takes 3.5uS on average to complete. When no prediction is being made, the circuit is power-gated. These overheads are included in all predictor results.

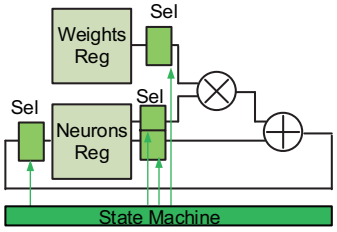

Figure 9. Neural networks computation serial architecture

5. RESULTS AND DISSICUSSION

In this section, we first examine the efficacy of the bottleneck resource predictor and smart frequency predictor, each in isolation, and then consider a system employing both techniques.

5.1 Bottleneck Resource Prediction Results

In the test, a home WiFi power profile is used as inputs. The testbenches are Mibench "large inputs".

Figure 10 and Figure 11 show the resource prediction results for two different testbenches on the same home WiFi power profile. The processor runs only in a portion of the total time. With more power available, the controller predicts to power on more resources to reduce the energy per instruction. When the input power is high, and the energy storage capacitor is full, the neural network controller predicts to power all the resources for the maximum forward progress, regardless of lower energy per instruction.

Different testbenches may require different configurations for best forward progress. If we compare testbench "rijndael_encoder" in Figure 11 to "susan_corners" in Figure 10, the "PRE" branch prediction is more likely to be the bottleneck resource, while the "ICL2" instruction cache level 2 is not.

The system provides a relatively high tolerance for prediction errors. Moreover, it is difficult to define an "absolute" error. For example, one good prediction is: utilizing the power income aggressively, then running with the minimum resources configuration at the next cycle. In the experiments, the predictor selects one configuration with the minimum resources at first and the rest of the energy is saved in the capacitor, then the predictor selects one configuration with the best energy per instruction for the next prediction cycle even if part of the energy has been leaked. Thus, the energy storage device provides a tolerance for prediction errors. As long as the forward progress is maximized, the predictor is still a good one.

Figure 12 shows the maximum forward progress improvement for different testbenches. Both Mibench and some neural network programs are tested. This method provides an average of 61.8% forward progress improvement. Forward progress improves for the following reasons: To begin with, when the input power is low, the predictor generates the minimum resources configuration for NVP to guarantee computation and to reduce the chance of backing up data to save power. Secondly, when the power supply exceeds a predefined power threshold, only the bottleneck resources are powered on. Thirdly, when the power is high and the stored energy level is full, all possible resources are powered on even if the energy per instruction is not the lowest.

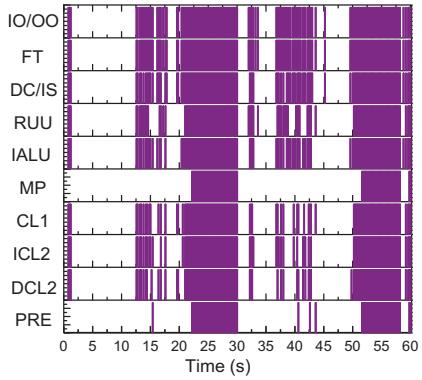

Figure 10. Testbench "susan_corners" resource allocation.

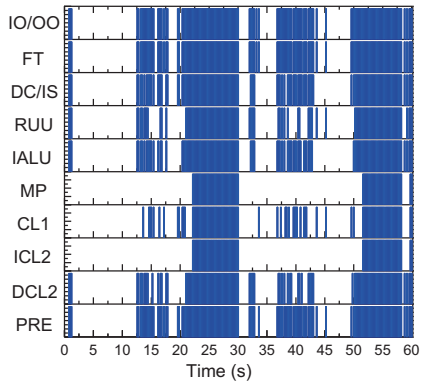

Figure 11. Testbench "rijndael_encoder" resource allocation.

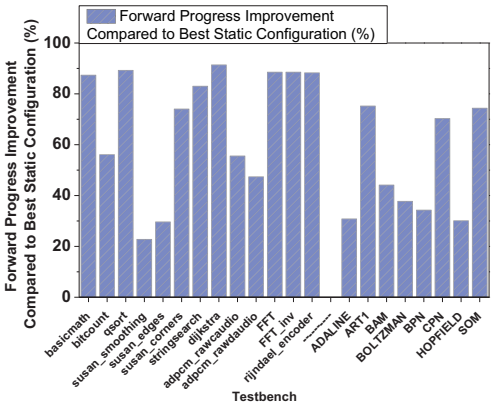

Figure 12. Bottleneck resource prediction: An average of 61.8% forward progress improvement.

5.2 Smart DFS Results and Analysis

When smart DFS is applied to the NVP, the system frequency dramatically changes with the input profile. As shown in Figure 13, the frequency almost follows the trends of the input power profiles. For a low input power and low stored energy, the smart DFS predictor uses a low frequency to reduce the activation threshold so as to aggressively use incoming energy, rather than storing it. For high power income scenarios, it bursts the frequency to a proper level that can just match the income power.

The capacitor does provide some buffering for the energy. Figure 13 shows two frequencies for two different testbenches. They have different energy per instruction when running on OoO NVP with fixed resources. This difference results in different frequency profiles. In the time section between 30s and 40s, the frequency for testbench HOPFIELD is higher than that of SOM. But between 40s and 50s, the frequency of SOM is higher than that of HOPFIELD. This indicates that current frequency and dissipated energy has influence on later prediction results through energy stored in capacitor. The capacitor is a cushion for improper frequency prediction because some of the energy can be saved to be used later, although with some leakage penalty.

Figure 14 is the forward progress improvement compared a best static frequency 672kHz, showing an average of 43.0% improvement. The variation among different testbenches is very small because the frequency changes only the energy used for forward progress computation, the EPI factor is offset during the computation for percentage improvement.

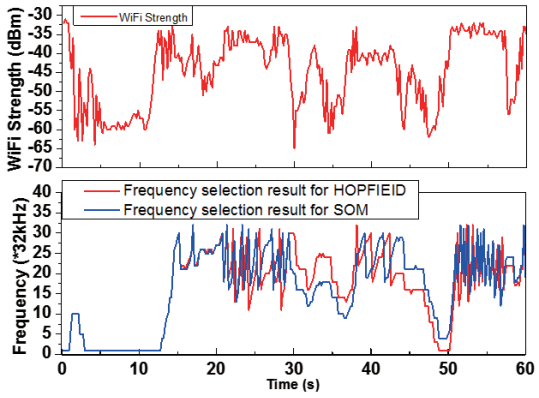

Figure 13. DFS frequency selection results.

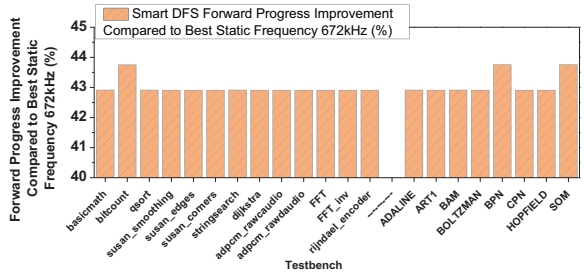

Figure 14. DFS frequency forward progress improvement compared to the best static frequency 672kHz

5.3 Resources Reallocation or DFS?

In its simplest form, we can think about forward progress via the following equation:

Forward Progress = Energy used for computation / Energy per Instruction (EPI) (1)

While backups and other overheads affect both terms on the right hand side, there is a clear intuitive mapping from each of our mechanisms to each of the right hand side terms: smart DFS primarily influences energy used for computation, and bottleneck resource prediction targets EPI. However, both approaches compete for the same income power to affect their benefits, and it is not immediately clear how best to apportion power between the two mechanisms.

Powering on all resources is rarely the most energy efficient way to use income energy. However, from the bottleneck resource predictor's perspective, in the absence of frequency scaling, aggressively using a large amount of temporary power income when the energy storage is full or almost full is still a good solution, even if the energy per instruction is not maintained at the best efficiency. When frequency scaling is also merged into the system, this situation changes: Targeting the best EPI point while bursting the frequency to aggressively use the income energy is the better solution at timescales large enough to support frequency boosting.

To support combined operation, we add two more modules to the Figure 8 dashed line box. These are needed because the changing resources make frequency prediction more complicated since different consumption levels occur for the same frequency setting. To avoid increased backups, once the stored energy level is less than 25%, the training module is triggered to compute the new weights, and update the weight in NVM, using one step lower frequency.

Figure 15 shows the results of the combined approach. One key behavior seen in Figure 15 is that, if the stored energy level is not full, the bottleneck resources predictor is unlikely to predict powering on all resources, and will continue at a more EPI-efficient point. The aggressiveness of the combined solution results in a smaller percentage of time that the stored energy is full compared to baseline, which helps keep the resource predictor operating in an EPI-sensitive region.

We observe that the EPI drops when more power is available due to powering on bottleneck resources, as shown in Figure 15. In contrast, when the input power is very small, the predictor generates minimum resource configurations with higher EPI to ensure the NVP continues running but avoids backup operations. The forward progress of the combined prediction scheme does fall short of what would be expected if the two techniques were orthogonal (i.e. 2.30X over baseline). This is because the bottleneck predictor still can either predict full resources or minimum resources, leading to a deviated EPI from the most efficient one.

Similar gains are seen across WiFi power traces. Across our benchmark suite, Spendthrift shows average forward progress improvements of, 2.09X, 1.94X, and 1.99X for each of 3 power profiles. Minimum and maximum improvements are 2.66X and, 1.76X, 2.16X and 1.84X, and 2.33X and 1.82X, respectively, for each of the three traces. Thus, the average improvement across all three traces for our suite is ~2X with an observed minimum of 1.76X over the best static baseline.

6. RELATED WORK

NVP architectures have been explored in previous works [1,2,4,28], in which NVP architecture trade-offs are considered. Noting the trade-offs among different microarchitecures, a machine learning method is proposed by Ma. et al. [3] to dynamically switch between three distinct design points. To the

best of our knowledge, there exist no works that attempt to predict bottleneck resources and DFS for NVP IoT platforms.

DFS is a traditional technique targeting either boosted performance or reduced energy [10-13]. Distinct from these works, we apply DFS to NVP for energy harvesting scenarios to adjust to variations in power-income rather than variations in the workload or thermal headroom. Traditional methods use lookup tables for DFS, while in more complex energy harvesting scenarios with unstable power, we apply machine learning to handle it.

Kontorinis, et al. [14] propose table-driven resource allocation in a configurable datapath to conservatively enforce peak power reductions with minimal performance degradation. We employ a similar scheme in this work to tune EPI to the current power income, but are not constrained to conservative solutions and utilize neural networks rather than tables to predict the bottleneck resources for unstable power incomes.

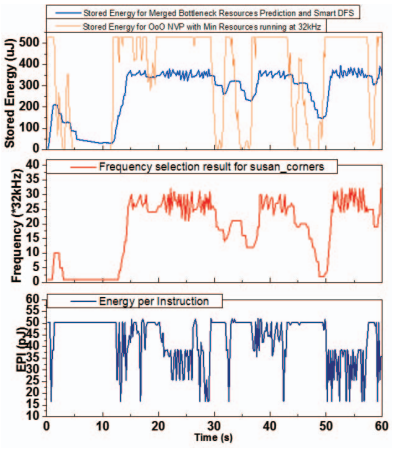

Figure 15. Fine-grained simulation results for merged bottleneck resources predictor and smart DFS.

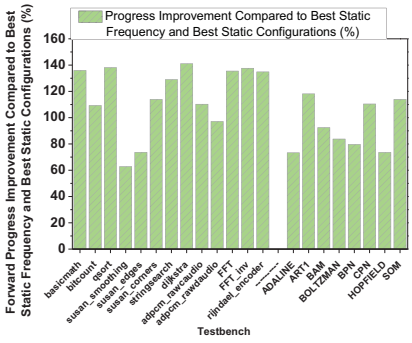

Figure 16. Forward progress improvement compared to OoO best static configuration and best static frequency. Average improvement of 2.08X.

7. CONCLUSION

This paper shows that for energy harvesting NVP, bottleneck resource prediction and frequency scaling can both be applied to improve the forward progress. Powering on some resources to mitigate some bottlenecks can actually significantly increase the NVP's parallel density. Through smart DFS, the energy used for forward progress computation increases. We show that the proposed spendthrift solution can achieve 2.08X forward progress than best case of OoO NVP with static configuration and static frequency.

REFERENCES

[1] Kaisheng Ma, et al., "Architecture exploration for ambient energy harvesting nonvolatile processors," HPCA'15 pp.526,537, 7-11 Feb. 2015

[2] Kaisheng Ma, et al., "Nonvolatile processor architecture exploration for energy harvesting applications", IEEE Micro, 35 (5), 32-40. 2015

[3] Kaisheng Ma, et al., "Dynamic machine learning based matching of nonvolatile processor microarchitecture to harvested energy profile. " ICCAD '15. 670-675. 2015

[4] Yongpan Liu, et al., "Ambient energy harvesting nonvolatile processors: from circuit to system." *DAC 2015*, pp.1,6, 8-12 June 2015

[5] Xueqing Li, et al., "RF-powered systems using steep-slope devices, "NEWCAS 2014, pp.73,76, 22-25 June 2014.

[6] Huichu Liu, et al., "Tunnel FET RF rectifier design for energy harvesting applications" JETCAS 2014 vol.4, no.4, pp.400,411, Dec. 2014.

[7] KaiSheng Ma, et al., "Nonvolatile processor optimization for ambient energy harvesting scenarios",NVMTS 2015, 2015.

[8] J. Charles et al., "Evaluation of the Intel® Core™ i7 Turbo Boost feature," IISWC 2009. pp. 188-197. 2009

[9] E. Rotem et al., "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," IEEE Micro, vol. 32, no. 2, pp. 20-27, March-April 2012.

[10] M. Bhat et al., "Implementation of dynamic voltage and frequency scaling for system level power reduction," I4C'14, pp.425-430. 2014

[11] B. Lin et al., "User- and process-driven dynamic voltage and frequency scaling," ISPASS 2009. pp. 11-22. 2009

[12] S. Park et al., "Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors," in TCAD, vol. 32, no. 5, pp. 695-708, May 2013.

[13] K. Choi et al., "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," in TCAD, vol. 24, no. 1, pp. 18-28, Jan. 2005.

[14] Vasileios Kontorinis, et al., "Reducing peak power with a table-driven adaptive processor core. ", MICRO 42, 189-200, 2009.

[15] D. Capalija et al., "Microarchitecture of a Coarse-Grain Out-of-Order Superscalar Processor," in IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 2, pp. 392-405, Feb. 2013.

[16] Matthew Guthaus, et al., "MiBench: A free, commercially representative embedded benchmark suite", IEEE 4th Annual Workshop on Workload Characterization, 2001.

[17] Karsten Kutza, http://baidu.nu/8sx43

[18] B. Widrow, et al., ""Adaptive Switching Circuit", IRE WESCON Convention Record, pp. 96-104, 1960

[19] G. Carpenter, et al., "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine", Computer Vision, Graphics, and Image Processing, 37, pp. 54-115, 1987

[20] B. Kosko. "Bidirectional Associative Memories", IEEE Transactions on Systems, Man, and Cybernetics, 18, pp. 49-60, 1988

[21] D. Ackley, et al., "A Learning Algorithm for Boltzmann Machines", Cognitive Science, 9, pp. 147-169, 1985

[22] D. Rumelhart, et al., "Learning Internal Representations by Error Propagation",Parallel Distributed Processing, Vol. 1, pp. 318-362, 1986

[23] R. Hecht-Nielsen, "Counterpropagation Networks", the IEEE International Conference on Neural Networks II, pp. 19-32, 1987

[24] J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proceedings of the National Academy of Sciences, 79, pp. 2554-2558, 1982

[25] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", Biological Cybernetics, 43, pp. 59-69, 1982

[26] Tom Schaul, et al., "PyBrain". Journal of Machine Learning Research, 2010.

[27] P. Royannez, et al., "90nm low leakage soc design techniques for wireless applications", ISSCC 2005, pp. 138–589 Vol. 1. 2005

[28] Kaisheng Ma et al., "Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power," in IEEE Micro, vol. 36, no. 3, pp. 72-83, May-June 2016.