A Performance and Power Co-optimization Approach for Modern Processors*

Yongxin ZHU
Dept. of Computer Science & School of Microelectronics
National University of Singapore & Shanghai Jiao Tong University
Singapore 117543, Singapore & Shanghai 200030, P.R.China
Email: zhuyx@comp.nus.edu.sg

Weng-Fai WONG
Dept. of Computer Science
National University of Singapore
Singapore 117543, Singapore
Email: wongwf@comp.nus.edu.sg

Cheng-kok KOH
Dept. of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-2035, USA
Email: chengkok@ecn.purdue.edu

Abstract

In embedded systems, performance and power are important inter-related issues that cannot be decoupled. Expensive and extensive simulations in a processor design space are usually required to verify whether a design meets both performance and power requirements. In this paper, an analytical co-optimization approach based on an integrated workload, performance and power model for modern processors is described and studied. A design space consisting of more than 15 architectural and workload parameters can be quickly explored for co-optimization. Validation with measured results obtained from simulators as well as physical processors showed that the model has a good degree of accuracy. We shall describe the details of approach and the model, and show how to apply the approach to the problem of co-optimizing the power and performance of processor design. With the completeness, flexibility and efficiency, our approach provides clear insights into the tradeoffs of designs for performance and power.

1 Introduction

In the era of mobile and pervasive computing, embedded systems are experiencing ever more complicated applications migrated from desktop platforms. The computation complexity of these applications requires higher computation capabilities that traditional scalar embedded processors may not offer. There are both academic [19] and industrial [8] attempts on increasing the computation performance by using superscalar architectures.

*This research was partially funded by A*STAR Grant 022/106/0043 and NUS Research Grant R-252-000-185-112.

Superscalar processing is the de-facto standard architecture for commercial off-the-shelf microprocessors. It is a complex scheme that involves the hardware dynamically issuing instructions. Modeling such processors is therefore also a complex task. However, there are numerous advantages of having a good model for such processors. Such a model can be used to design future processors as well as to gain insight into the behavior of applications under superscalar processing.

The modeling methodology we have taken includes the usual five steps of performance modeling, namely trace collection, separable components analysis, modeling and measurement of machine behavior, performance validation, and model validation [3]. Our work resembles that of Austin and Sohi [2], Lam and Wilson [13], Dubey et. al. [6], and Noonburg and Shen [15] in the use of similar separable components. Part of our model, i.e. the modeling of the central (issue) window, is built on the work of Pyun et. al. [17]. We went beyond their work by having a comprehensive model that accounts for all the key components of modern superscalar processors.

Austin and Sohi [2] used dynamic dependence graphs to expose the parallelism in program traces. The studies of Lam and Wilson [13], and Dubey et. al. [6] were based on probability theory and experimental simulations. One shortcoming of these models is that they do not capture detailed dependency metrics accounting for architectural features, such as out-of-order issue. Nonetheless, these studies generally formed the bases of analytical models of superscalar processors. Noonburg and Shen [15] accounted for program parallelism and machine parallelism by representing dependencies and branch distributions in terms of probability vectors. The caches, retirement rate, and multiple-cycle functional units are not considered in their model. Pyun et. al. [17] quantified the relationship among instruction issue policy, dependencies in the instruction queue and func-

tional units. The model however, did not distinguish different classes of functional units. Karkhanis and Smith [9] recently proposed a first-order performance model framework based on miss events. An "oldest-first" priority scheme is assumed for instruction issue, which does not model dependency resolution clearly.

In a previous work [22], a queuing model is presented for superscalar processors based on a network of Multiple Class and Multiple Resource (MCMR) queues. Using this model, three classes of architecture modifications were studied through sensitivity analysis [23, 24]. Sensitivity analysis gives qualitative insight into the nature of the performance improvement with respect to these architecture changes, thereby assessing the merits of these changes. The more recent paper [25] further extended the previous works to consider out-of-order issue processors. Moreover, the model was inclusive of processor power analysis.

As far as we know, there are very few works focusing on co-optimization of processor power and performance. A possibly relevant one is done by Nakamura et al. [14], where compiler technologies were employed to reduce memory accesses, consequently improve performance and save memory power.

In this paper, we proposed a co-optimization work flow based on the integrated power and performance model to address the problem of co-optimization for both performance and power consumption. The work flow allows both partial and full parameter re-extraction during the optimization process. Its efficiency and completeness make it a useful method to explore huge design space of embedded processors under power constraints.

We shall brief our performance model in Section II, our power model in Section III, and validations in Section IV. Section V will describe the co-optimization approach and examples of applications of the approach. That will be followed by a conclusion.

2 Performance Model

To model a generic superscalar processor, we used a *network* of *multiple-class multiple-resource* (MCMR) systems. Each stage of the pipelines contributes to the final results of the processor. The lowest throughput of all the pipeline stages is the bottleneck of the entire processor and determines the maximum possible throughput of the processor. We shall now recall the main results of the performance model [24, 25].

The throughput of the processor Θ is the minimum of the service rates of *decoder unit* (μ_{dec}), *central window* (μ_{win}), and *retirement unit* (μ_{ret}):

$$\Theta = \min\{\mu_{dec}, \mu_{ret}, \mu_{win}\} . \quad (1)$$

μ_{dec} denotes the average decoding rate without overflow in the central window. Let W_{dec} denote the decoding width; $\overline{I}_{br}$ be the average number of (non-branch) instructions between two branch instructions; T_{br} be the misprediction penalty time; $p_{ins,miss}$ be the instruction cache miss ratio; $t_{ins,pen}$ be the instruction cache miss penalty time; and $p_{br,prtd}$ be the probability of a correct branch prediction. If $\overline{I}_{br} < W_{dec}$,

$$\mu_{dec} = \frac{C_1}{C_2 + C_3 \times t_{ins,pen} \times p_{ins,miss}} , \quad (2)$$

where C_1, C_2, and C_3 are linear functions of $\overline{I}_{br}, T_{br}, W_{dec}$, and $p_{br,prdt}$. The rest cases can be looked up in [24].

μ_{ret} denotes the retirement rate under an in-order retirement policy. Its parameters are as follows. W_{ret} is the the maximum number of instructions that can be retired per cycle. $\overline{D}$ is the average dependence distance (inclusive of one of the instruction in the dependence) between two instructions that have a data dependence relation. For $\overline{D} < W_{ret}$, μ_{ret} is given below, the rest cases can be referred in [24]:

$$\mu_{ret} = (2 \times \overline{D})/(1 + T_{dep}) , \quad (3)$$

where T_{dep}, the average time for an antecedent instruction to pass through the functional units, is:

$$T_{dep} = [\sum_{i}^{type} (t_i \times S_i)] \times (1 + \overline{P}_{dep}) , \quad (4)$$

where $type \in \{ieu, fpu, lsu, br\}$ is the set of types of functional units, namely the integer execution unit, the floating point unit, the load store unit and the branch unit. $S_i \in [0, 1]$ is the fraction of the total number of instructions that is executed on functional unit i, and t_i is the average service time of each functional unit of type i. Typically, $t_{ieu} \in \{1, 2\}$, $t_{fpu} \in \{3, ..., 6\}$, $t_{lsu} = p_{d,prtd} + t_{dat,pen} \times (1 - p_{d,prtd})$ and $t_{br} = p_{i,prtd} + t_{ins,pen} \times (1 - p_{i,prtd})$. The parameters $p_{d,prtd}, p_{i,prtd} \in [0, 1]$ represent the probabilities of the data cache prediction and the instruction cache prediction, respectively.

For μ_{win}, we consider out-of-order issue processors with multiple instruction types or classes. On out-of-order-issue processors, any independent and ready instruction in the instruction window may be dispatched to an available functional unit. Hence, μ_{win} is the total sum of service rates of functional units μ_t where t is an instruction type, that is $\mu_{win} = \mu_{ieu} + \mu_{fpu} + \mu_{lsu} + \mu_{br}$.

Given $\rho_{k,t}(Z_{win})$ as the probability that k instructions of type t are issued from the window of size Z_{win} and F_t representing the number of functional units of type t. then:

$$\mu_{win} = \sum_{t}^{type} \sum_{k=1}^{F_t} (\rho_{k,t}(Z_{win}) \times k) , \quad (5)$$

and $\rho_{k,t}(Z_{win}) = \mathcal{P}_{k,t}(Z_{win}) \times \phi_{pipe,t}(k)$, where $\mathcal{P}_{k,t}(Z_{win})$ is the probability that k independent instructions are extracted from Z_{win} instructions and $\phi_{pipe,t}(k)$ is the probability that at least k pipeline units of type t are available. Detailed discussions on $\mathcal{P}_{k,t}(Z_{win})$ and $\phi_{pipe,t}(k)$ can be found in [25].

Tech.	stat. pwr. /tot. pwr. without leakage opti.	stat. pwr./tot. pwr. with leakage opti.	V_{dd}
0.35μm	9.8%	6.6%	3.3
0.18μm	22.6%	11.7%	1.8
0.13μm	43.4%	26.9%	1.5
0.10μm	48.1%	25.5%	1.2
0.07μm	56.2%	25.1%	0.9

Table 1. Prop. of leak. power in tot. power.

3 Power Model

The power consumption of a resource consists of a dynamic and a static component, i.e., $\pi_{tot,res} = \pi_{static,res} + \pi_{dyn,res}$. The static portion is given by $\pi_{static,res} = I_{static,res} \times V_{dd}$. The leakage current $I_{static,res}$ is an exponential function of threshold voltage V_t (in mV) by Sylvester and Keutzer [20]. For any technology node, the static power takes a usually stable portion of the total power. Khouri and Jha [10] summarized the ratios of the static power over the total power based on 6 different circuits. We use the averaged ratios in TABLE I.

For the dynamic power component, we model dynamic power as a traditional function of *dynamic capacitance* ($\mathcal{C}_{res}$), the *supply voltage* (V_{dd}) and the *clock frequency* (Ω):

$$\pi_{dyn,res} = \mathcal{C}_{res} \times V_{dd}^2 \times \Omega \ . \tag{6}$$

The accesses to each resource are obtained from the simulators. With total dynamic capacitance and number of accesses of a resource, we can obtain the *dynamic capacitance per access to the resource* ($\mathcal{C}_{a,res}$) for each benchmark. This enables us to establish a link between the performance model and the power model. We also need the *average number of accesses to the resource per request (instruction)*, denoted by $N_{a,req,res}$. Then the *number of accesses a resource services each cycle* $N_{a,res}$ can be obtained as: $N_{a,res} = \mu_{res} \times N_{a,req,res}$.

The link to power is expressed as the *dynamic capacitance per resource per cycle*, $\mathcal{C}_{res,cycle} = N_{a,res} \times \mathcal{C}_{a,res}$. We assume $\mathcal{C}_{res}$ to be equal to $\mathcal{C}_{res,cycle}$, so $\mathcal{C}_{res} = \mu_{res} \times N_{a,req,res} \times \mathcal{C}_{a,res}$. With the total dynamic capacitance per resource or $\mathcal{C}_{res}$, we can obtain the power in 6. Explanation on how to obtain μ_{res} from the performance model can be found in [25]. The dynamic power costs of all resources form the total dynamic power:

$$\begin{aligned}\pi_{dyn,tot} = \pi_{win} + \pi_{ret} + \pi_{dec} + \pi_{ieu} + \pi_{fpu} \\ + \pi_{lsu} + \pi_{br} + \pi_{icache} + \pi_{dcache} \ .\end{aligned} \tag{7}$$

4 Validations of the Model

For the integrated model, extensive validations against measured results on simulators and actual processors showed a high degree of accuracy. The performance model attained average relative errors of 5.08% [22–24], and 5.31% [25] compared with an in-order-issue UltraSPARC-I processor, and the out-of-order-issue SimpleScalar simulator. In this paper, we extend the validation to a physical processor UltraSPARC-III [21]. Most of parameters for our model were obtained from UltraSPARC-III processor's hardware performance counters. The dependency metrics were counted with additional profiling using the PAPI library [16]. With a physical processor, it is very difficult to tell where our model lost accuracy. Nonetheless, as far as we know, the 17% relative error is the best, if not the only, validation results for a theoretical model compared against a physical processor ever reported.

The power model achieved an average relative error of 10.9% [25] when compared to full simulation. Bose et. al. [4] pointed out that high-level models (such as those based on instruction traces) often lack the fidelity necessary to accurately reproduce the complex nature of data-dependent micro-architecture optimizations. We believe that our model is a step in making this possible.

5 Power and performance Co-optimization

Having modeled the power and performance in an integrated manner, we can evaluate architecture configurations under constraints of both performance and power. We start with the details on how we extract the relevant parameters for the co-optimization.

5.1 Parameter Extraction

To obtain the parameters, we use stand-alone profiling tools and analyzers attached to the benchmarks running on a physical processor or simulator. For example, the profiling tool for the simulated SimpleScalar processor is "sim-profile". We also embedded our analyzer into the "sim-outorder" simulator.

Most of the parameters for the model can be measured quickly using the profiling technique. These parameters are S_{ieu},S_{fpu}, S_{br}, S_{lsu}, $\overline{I}_{br}$, $\overline{D}$, $\overline{P}_{dep}$, and p.

S_{ieu},S_{fpu}, S_{br}, S_{lsu} are obtained by counting instructions in different categories and the total number of instructions. $\overline{I}_{br}$ is obtained by totalling the distance between two most adjacent branch instructions. To find $\overline{D}$, we count the instances of dependencies for each type of dependencies. We also count the distance between an instruction and its dependent ancestor instruction for each type of dependencies. The number of instances of dependencies are also used to obtain $\overline{P}_{dep}$ and p. Other parameters, q, $p_{ins,miss}$, $p_{dat,miss}$ and $p_{br,prtd}$, are collected during the simulations of benchmarks.

To calculate q accurately, we need to check in each simulation cycle if a ready instruction, whose dependencies are resolved, can be given an available resource for execution. To approximate q, we just simply slide an analysis window, whose width is the issue width, through the profiled code. For each instruction class in the window, if the number of non-dependent instructions is less than the number of resources for the class, then we assume these instructions can

be issued. The number of issued instructions along with the total number of instructions give us q.

We can approximate the miss ratios $p_{ins,miss}$, $p_{dat,miss}$ by looking up the miss ratios from Cantin and Hill's results [5]. These ratios can also be estimated with additional analytical cache models [7]. Alternatively, a single-pass, multiple configuration cache simulation [18] such as the Cheetah cache simulator that comes with SimpleScalar simulator can be used to facilitate the collection of cache miss ratios over a broader range of cache configurations in a single simulation. We further assumed $p_{br,prtd}$ is equivalent to $p_{ins,miss}$. Table 2 lists the extraction time for the benchmarks `183.equake` with a test input set, and `bzip` with a reference input set on a simulated SimpleScalar processor. The input parameters are listed in Table 3 and Table 4.

parameter set	extract. time (s) of equake	extract. time (s) of bzip
full parameters	3778	358656
S_{ieu},S_{fpu}, S_{br}, S_{lsu},$\overline{I}_{br}$, p, $\overline{D}$,$\overline{P}_{dep}$, p	197	14985
q, $p_{ins,miss}$, $p_{dat,miss}$,$p_{br,prtd}$	3581	343671

Table 2. Parameter extraction time.

Benchmark	bzip2	equake	gzip	mcf	mesa	vpr
IEU1-4	45.7%	26.3 %	46.8 %	39.4 %	42.2 %	43.6 %
FPU1-4	0.0%	15.3%	0.0 %	0.0 %	7.0 %	5.6 %
BRU	15.9%	6.1%	19.6 %	2.7 %	1.0 %	1.0 %
LSU	28.5%	41.5%	23.8 %	48.2%	53.4 %	53.4 %
$\overline{D}$	1.996	1.955	1.983	2.016	1.873	1.911
$\overline{I}_{br}$	7.26	6.69	4.38	3.65	4.18	4.74
$\overline{P}_{dep}$	0.562	0.504	0.627	0.620	0.425	0.589
T_{dep}	1.972	2.403	1.985	2.085	2.248	2.187
p	0.438	0.4962	0.3729	0.3802	0.5755	0.5178
q	0.991	0.975	0.997	0.995	0.95	0.983
$p_{ins,,miss}$	0.0110	0.0343	0.0008	0.0038	0.0296	0.0067
$p_{dat,miss}$	0.0227	0.0552	0.0603	0.1589	0.0221	0.0459

Table 3. Characteristics of the benchmarks used as inputs to our Model.

5.2 Solving the Co-optimization Problem

To co-optimize power and performance, we have to minimize $\pi_{dyn,tot}$ in Eq.(10), while maximizing the throughput in terms of number of instructions per second, i.e. $\Theta \times \Omega$. Firstly, we shall assume that the user sets an upper limit, π_U say, i.e. $\pi_{dyn,tot} \leq \pi_U$.

Within this constraint, we seek to maximize Θ in Eq. 1 along with varying Ω. In short, we would like to maximize throughout under a power budget.

In order to find the configuration with the least energy consumption for a computation, we look for the minimal total energy to complete the task of executing n_i instructions.

Bench.	bzip2	equake	mcf	mesa	vpr
$\mathcal{C}_{a,win}$	0.631	0.898	1.004	0.762	0.769
$\mathcal{C}_{a,regfile}$	2.665	3.806	4.527	3.330	3.590
$\mathcal{C}_{a,dec}$	0.421	0.603	0.614	0.485	0.501
$\mathcal{C}_{a,ieu}$	16.32	24.09	26.18	19.56	20.33
$\mathcal{C}_{a,fpu}$	16.32	24.09	26.18	19.56	20.33
$\mathcal{C}_{a,lsu}$	2.527	3.981	4.087	3.912	3.035
$\mathcal{C}_{a,br}$	38.90	53.14	37.39	28.84	43.83
$\mathcal{C}_{a,icache}$	2.751	3.911	3.846	3.152	3.194
$\mathcal{C}_{a,dcache}$	17.09	27.02	27.72	27.35	24.33

Table 4. Capacitance (in farad) related parameters as inputs to our power model.

Let $\pi_{u,x}$ be the upper bound for power for x, a solution point in the space of feasible configurations. The constraint $\pi_{dyn,tot} \leq \pi_{u,x} \leq \pi_U$ must hold as we are searching for the maximum performance $\Theta \times \Omega$.

If such a point x in the configuration space exists, then the time to execute the application is $n_i/(\Theta \times \Omega)$. Consequently, this will also yield the minimal total energy, where the power is $\pi_{dyn,tot}$:

$$E_x = n_i \times \pi_{dyn,tot}/(\Theta \times \Omega). \quad (8)$$

5.3 Co-optimization Work Flow

The work flow with full parameter extraction is illustrated in the left part of Fig. 1. It starts with the parameter extraction which can be done either during an application's execution or simulation, or from stored trace files. These parameters are fed into the model to calculate performance or power metrics. The next stage is to judge if these metrics meet the design constraints on the performance or power. Once the constraints are satisfied, the flow completes. Otherwise, some of the parameters should be modified by enumerating. The next stage is chosen according to whether the modified parameters affect other parameters extracted. If it does, the next stage should be partial or complete rerun of the extraction process. Otherwise, the next stage is the analytical calculation.

The model parameters that do not change during the optimization process are W_{dec}, W_{ret}, t_{br}, $t_{ins,pen}$, $t_{dat,pen}$, t_{ieu}, t_{fpu}, $p_{br,prtd}$, $\overline{P}_{dep}$, and p. The remaining ones, namely, W_{win}, q, $p_{ins,miss}$, F_{ieu}, F_{fpu}, F_{lsu}, and F_{br} would require re-extraction if the configuration changes. The latter parameters can be approximated by a rerun of profiling. For run-time efficiency, we can construct a look-up table and use interpolation and/or extrapolation to estimate these parameters. The approach we take here is that of re-profiling by looking up tables of reference values or a rerun of profiling. This approximation forms the another flow shown in the right part of Fig. 1. For example, $p_{ins,miss}$ can be found from Cantin and Hill's results [5]. F_{ieu}, F_{fpu}, F_{lsu}, and F_{br} are taken into consideration to calculate q during the profiling run. To determine $\overline{P}_{dep}$ and p, W_{win} is

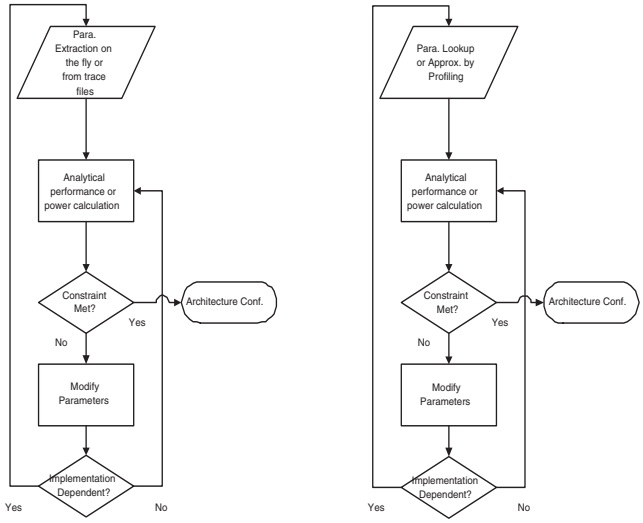

Figure 1. Work flows with full parameter extraction or approximate parameter lookup.

considered during the quick profiling.

It is noteworthy that it is difficult to use a physical processor to perform the optimization. In the process of optimization, changes are made to the configuration of the processor. Therefore, modeling and simulation are the only two choices. Of these two options, our model allows for the immediate enumeration of the design subspace consisting of the first set of parameters mentioned above that do not require re-extraction. That same subspace would require thousands of simulation runs. To explore a design space containing both sets of parameters, a partial re-extraction of parameters is required if the optimization process loops on the affecting parameter. In this case, the model's exploration is still much faster since it takes almost no time to explore the subspace containing the first set of parameters.

Conf. Changes	Anal. Power (watts)	Para. Re-extract. Time (s)	Modelling Time(s)	Simu. Power (watts)	Simu. Time (s)
orig. config.	31.10	0	2	32.85	4186
double W_{dec}	36.26	0	2	39.85	4186
reduce L1 inst. cache to 64 lines	28.45	3581	2	29.02	4186

Table 5. Costs in Optimization Process.

Table 5 illustrates the costs of an optimization process with the benchmark `183.equake`. W_{dec} is a parameter that does not require re-extraction while changes to the L1 instruction cache will impact other parameters. While the latter requires a partial re-extraction of parameters, the cost is still less than the simulation cost.

5.4 Applications of the Approach

We shall demonstrate the usability and flexibility of the approach with examples to solve different problems.

Optimizing Clock Frequency: We will now use `bzip2` as an example to show how co-optimization is achieved. To begin, we set an upper bound on the dynamic power, $\pi_{dyn,U} = 25$ watts. The co-optimized solution is obtained by the following search procedure:

1. Read the performance values of `256.bzip2` from Table 3 : $\{S_{ieu} = 0.457, S_{fpu} = 0.0, S_{lsu} = 0.285, S_{br} = 0.159, \overline{D} = 1.9960, \overline{I}_{br} = 7.26, \overline{p}_{dep} = 0.562, p_{ins,miss} = 0.0110, p_{dat,miss} = 0.0227, p_{d,prtd} = 1 - p_{dat,miss}, p_{i,prtd} = 1 - p_{ins,miss}, p_{br,prtd} = 1 - p_{ins,miss}\}$. These benchmark specific parameters along with architectural parameters $\{t_{ieu} = 1, t_{fpu} = 3, t_{dat,pen} = 3, t_{ins,pen} = 2, Z_{win} = 8, type = 4, W_{dec} = 4\}$ are fed into (2), (3) and (5) to obtain μ_{dec}, μ_{ret}, and μ_{win} then $\mu_{lsu} = \mu_{dec} \times S_{lsu}$, μ_{icache} and μ_{dcache}.

2. For the power constraint on the dynamic power, $\pi_{u,x}$ from 25 watts down to 1 watt in steps of -1 watt do:

   2.1. For each clock frequency Ω from 100 to 600 MHz at a step of 100 MHz, we repeat the following steps to obtain the maximum performance $\Theta \times \Omega$ under the power constraint of 25 watts.

   2.1.1. With the above performance service ratios of resources and Ω, we obtain π_{res} in (9), where $\mathcal{C}_{a,res}$ is obtained from Table 4.

   2.1.2. Sum up π_{res} for all components. If the total $\pi_{dyn,tot}$ is less than π_u, then we have found a configuration within the constraints. We also note down the performance $\Theta \times \Omega$ and $\pi_{dyn,tot}$.

3. Find the maximum of $\Theta \times \Omega_i$ and its associated $\pi_{dyn,tot}$ and Ω_i.

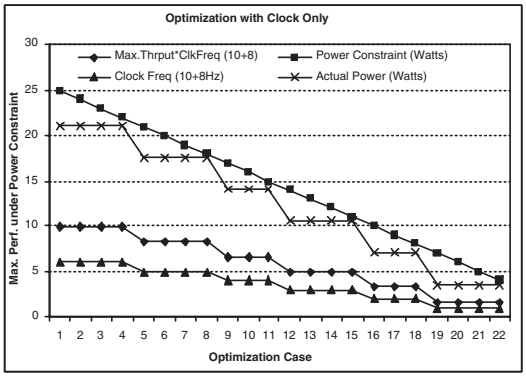

Figure 2. Optimal performance by varying only clock within power constraints.

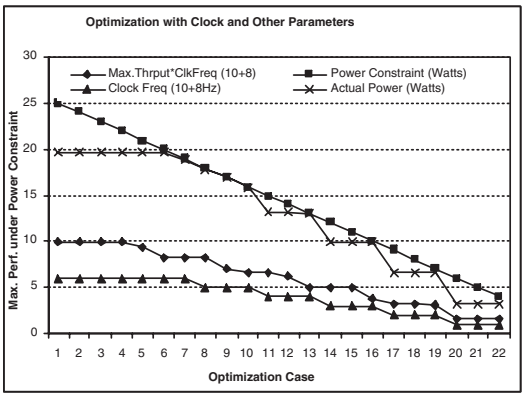

Figure 3. Optimal performance by varying clock and others within power constraints.

The X axis in Figure 2 and Figure 3 indicates different optimization 'cases' where variant power constraint, and a combination of optimization choices. In Figure 3, in addition to clock frequency, we tried varying a number of other parameters including t_{ieu}, t_{fpu}, T_{br}, W_{dec}, $t_{dat,pen}$, $t_{ins,pen}$, $p_{ins,miss}$, $p_{d,prtd}$, $p_{i,prtd}$ and p. Figure 2 only shows the varying clock with all others constant.

For example, the first optimization case in Figure 2 shows the impact of varying only the processor clock on the optimization. This first case includes a power constraint $\pi_{u,x}$ of 25 watts, clock frequencies range of 100 to 600MHz and other fixed parameters regarding the configuration and the benchmark. The Y axis shows the product of maximum throughput and clock frequency under power constraint.

The results show the dominance of clock frequency in the determination of optimal solution. By varying the clock frequency alone within the range of 100 to 600 MHz, we are able to obtain optimal $\Theta \times \Omega$ until the power limit $\pi_{u,x}$ becomes less than 3 watts (the 22nd case in Figure 2). There is no solution beyond this $\pi_{u,x}$. The results indicate the clock frequency alone in the range of 100 to 600 MHz is effective to lower the power consumption till the constraint of 7 watts in the 22nd case. Figure 3 shows the co-optimization based on the combination of clock frequency and 10 architectural parameters can further improve the maximum throughput in a constantly declining curve. Besides, the variations of other architectural parameters together with the clock frequency produce about a 7% higher maximum $\Theta \times \Omega$ than the clock alone satisfying all power constraints. This is due to the fact that the variations allow for a higher actual power consumption, which is closer and yet still below the power limit, a higher throughput can be obtained. As in the case of process technology, supply voltage by itself does not affect the throughput as it is not involved in the calculation of the throughput.

Minimizing Energy by Changing Level 1 Instruction Cache: We now use `183.equake` as an example to illustrate how the configuration with the minimal total energy can be found. The benchmark `183.equake` has a total of 1.3691×10^9 instructions with the test input set. We set dynamic power limits $\pi_{u,x}$ to be less than 33 watts and Ω=600MHz. We explore design subspace in which the number of lines in a direct mapped L1 instruction cache, whose line width is 32 words, is varied while the other parameters remains unchanged. Here, the cache miss ratios are obtained through simulation run although it would be possible to use other analytical cache models [12] to obtain estimates. There are five solutions listed in Table 6 in

#lin	$p_{ins,miss}$	E_x	$E_x^{'}$	π_{tot}	$\pi_{u,x}$	**max.**($\Theta \times \Omega$)
64	10.63%	62.70	62.57	26.56	27	7.799×10^8
128	6.86%	47.64	47.97	27.16	28	7.805×10^8
256	3.11%	49.58	50.80	28.43	29	7.850×10^8
512	0.48%	53.45	56.29	30.78	31	7.882×10^8
1024	0.38%	53.72	56.65	30.93	31	7.883×10^8

Table 6. The impact of L1 instruction cache on total energy.

which the second solution has the minimum of 47.64 joules. In other words, total energy is minimized with a L1 instruction cache of 128 lines. In Table 6, **#lin,** $p_{ins,miss}$**,** E_x**,** $E_x^{'}$**,** π_{tot}**, max.**($\Theta \times \Omega$), stand for the number of lines, the miss ratios, analytical energy (in joules), simulation energy (in joules), analytical power (in watts), and maximum $\Theta \times \Omega$, respectively.

Other than our analytical method, a brute force method by simulation takes more than 20900 seconds to obtain these above results for the benchmark `183.equake` given the test input set. The method even takes much more time given a longer input set such as the reference input set.

Impact of Leakage Power on the Optimization: As the feature size decreases with scaling down technologies, the leakage power starts to account for a significant portion of the total power budget. Following our approach, we can illustrate the impact of leakage power on the maximum clock frequencies and dynamic power consumptions. The results are shown in Fig. 4. The proportions of the leakage power in the total power are reported by Khouri and Jha [10]. In this showcase, we varied the clock frequency while keeping the rest of the parameters unchanged.

Fig. 4 shows that the leakage power has a more significant impact on $130nm$ technology nodes than 180nm. For 130nm technology nodes without optimization on the leakage power, the maximum clock frequency allowed is 1.4GHz. Once the leakage power is improved as in [10], the clock frequency allowed is extended to 1.8Ghz. In comparison, the 180nm technology nodes only see an extension from 1.3Ghz to 1.5Ghz.

6 Conclusion

In this paper, we proposed a co-optimization approach to both performance and power issues based on a unified analytical model that accounts for both power and performance.

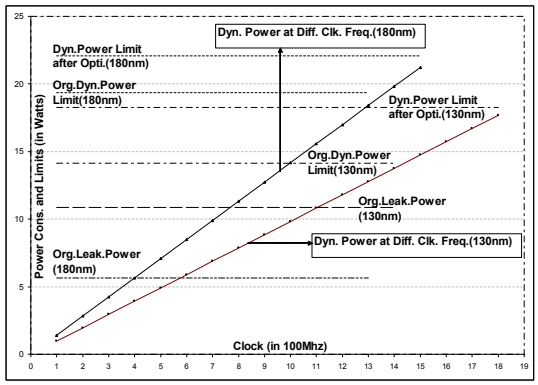

Figure 4. Power variations constrained by tot. power of 25W with 130nm & 180nm tech.

That model stands extension validations on simulators and physical processors.

In our co-optimization process, the costs to extract parameters for our model are low as instruction traces are analyzed once to obtain a set of key parameters which characterize the traces. We showed how the approach works to quickly explore large portions of the design space and co-optimize both power and performance. With this approach, searching for the processor configuration that yields the minimum energy under constraints becomes doable.

As demonstrated in the exercises on clock settings, leakage power and cache configuration optimization, the co-optimization approach revealed interesting insights that are consistent with conventional wisdom and experiences in processor design. We believe our approach will be a useful tool for designers to handle the dilemma between satisfying computation demands and power budgets.

References

[1] Austin, T., Burger, D., Sohi, G., Franklin, M., Breach, S., and Skadron, K., 'The SimpleScalar Architectural Research Tool Set.' [Online]. Available: *http://www.cs.wisc.edu/ mscalar/simplescalar.html*, 1998

[2] Austin, T. M., and Sohi, G. S. 'Dynamic Dependency Analysis of Ordinary Programs', *Proc. 19th Int'l Symp. on Comp. Arch.*, pp. 342-351, 1992.

[3] Bose, P., and Conte, T. M., 'Performance Analysis and Its Impact on Design', *Computer*, Vol. 31, No.5, pp.41-49, May, 1998.

[4] Bose, P., Conte, T. M., and Austin, T.M., 'Challenges in processor modeling and validation', *Micro, IEEE*, Vol. 19, No.3, pp.9-14, May,1999.

[5] Cantin, J. F. and Hill, M. D., 'Cache Performance for SPEC CPU2000 Benchmarks, Version 3.0', *Available online, http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/*, 2003

[6] Dubey, P. K., Adams, G. B., and Flynn, M. J., 'Instruction Window Size Trade-Offs and Characterization of Program Parallelism', *IEEE Trans. on Comp.*, 43(4), pp. 431-442, 1994.

[7] Hennessy, J.L., and Patterson, D.A., *Computer Architecture: A Quantitative Approach,* (3rd ed.), Morgan-Kaufmann, San Francisco, USA, 2003.

[8] Intel Incorporation, ' Intel i960 Processors', *online: http://www.intel.com/design/support/faq/i960_processors/i960_general.htm*

[9] Karkhanis, T.S. and Smith, J.K., 'A first-order superscalar processor model', *Proc. of 31st Int'l Symp. on Comp. Arch.(ISCA)*, pp. 338-348, Jun. 2004.

[10] Khouri, K.S. and Jha, N.K., 'Leakage Power Analysis and Reduction During Behavioral Synthesis', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 10, Number 6, pp. 876-885, Dec. 2002.

[11] Kleinrock, L., *Queueing Systems* (vols. 1 & 2), John Wiley & Sons, Inc. New York, USA, 1975

[12] Li, H., Bhunia, S., Chen, Y., Roy, K. and Vijaykumar, T.N.,'DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design', *VLSI Systems, IEEE Transactions on*, Vol. 12 , Issue 3, pp. 245 - 254, March 2004.

[13] Lam, M. S., and Wilson, R. P., 'Limits of Control Flow on Parallelism', *Proc. of 19th Int'l Symp. on Comp. Arch(ISCA)*, pp. 46-57, 1992

[14] Nakamura, H. and Kondo, M. and Ohneda, T. and Fujita, M. and Chiba, S. and Sato, M. and Boku T., 'Architecture and Compiler Co-Optimization for High Performance Computing', *Proc. of IWIA'02*, Jan., 2002

[15] Noonburg, D. B., and Shen, J. P., 'Theoretical Modeling of Superscalar Processor Performance', *Proc. of MICRO 27*, pp. 53-62, San Jose, USA, 1994

[16] The PAPI Team, 'The Performance Application Programming Interface.' [Online]. Available: *http://icl.cs.utk.edu/papi/index.html*, 2004

[17] Pyun, Y.H., Park, C.S., Choi, S.B., 'The Effect of Instruction Window on the Performance of Superscalar Processors', *IEICE Trans. on Fundamentals of Electronics Communications and Computer Sciences*, E81A: (6), pp. 1036-1044, Jun. 1998.

[18] Sugumar, R.A. and Abraham S.A., 'Set-associative Cache Simulation Using Generalized Binomial Trees', *ACM Transactions on Computer Systems (TOCS)*, Vol. 13 , Issue 1, pp. 32-56, Nov. 1995.

[19] Suzuki, K and Arai, T. and Nadehara, N. and Kuroda, I. 'V830R/AV: Embedded Multimedia Superscalar RISC Processor', *IEEE Micro*, Volume 18 , Issue 2, pp. 36 - 47, March, 1998.

[20] Sylvester, D. and Keutzer, K., 'Getting to the Bottom of Deep Submicron', *Proc. of Int'l Conference on CAD*, pp. 203-211, Nov. 1998.

[21] Sun Micrsystem Corporation, 'UltraSPARC III Cu Processor', *Available online, http://www.sun.com/processors/UltraSPARC-III/DS00101USIIICu0902.pdf*, 2001.

[22] Zhu, Y., and Wong, W. F., 'The Effect of Instruction Dependency on Superscalar Processor Performance' *Australian Computer Science Communications*, pp. 215-226, Volume 20, Number 4, Springer-Verlag, 1998

[23] Zhu, Y., and Wong, W. F., 'Modeling Archietctural Improvements in Superscalar Processors', *Proc. of HPC-Asia 2000* pp. 28-30. Beijing, P.R.C., 2000

[24] Zhu, Y., and Wong, W. F., 'Sensitivity Analysis of a Superscalar Processor Model' *Proc. of the Seventh Asia-Pacific Computer Systems Architectures Conference*, Melbourne, Australia, pp. 109-118. Jan. 2002.

[25] Zhu, Yongxin, and Wong, Weng-Fai, and Andrei, Stefan, 'An Integrated Performance and Power Model For Superscalar Processor Designs', *Proc. of IEEE ASP-DAC 2005*, Shanghai, P.R.C, 2005