PROCEEDING OF THE 1995 ANNUAL
28th HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (Maui, HI)
(winner: best paper)

# A Technique to Determine Power-Efficient, High-Performance Superscalar Processors

Thomas M. Conte Kishore N. P. Menezes Sumedh W. Sathaye
Computer Architecture Research Laboratory
Department of Electrical and Computer Engineering
University of South Carolina
Columbia, South Carolina 29208

**Abstract**

Processor performance advances are increasingly inhibited by limitations in thermal power dissipation. Part of the problem is the lack of architectural power estimates before implementation. Although high-performance designs exist that dissipate low power, the method for finding these designs has been through trial-and-error. This paper presents systematic techniques to find low-power, high-performance superscalar processors tailored to specific user benchmarks. The model of power is novel because it separates power into architectural and technology components. The architectural component is found via trace-driven simulation, which also produces performance estimates. An example technology model is presented that estimates the technology component, along with critical delay time and real estate usage. This model is based on case studies of actual designs. It is used to solve an important problem: increasing the duplication in superscalar execution units without excessive power consumption. Results are presented from runs using simulated annealing to maximize processor performance subject to power and area constraints.

The major contributions of this paper are the separation of architectural and technology components of dynamic power, the use of trace-driven simulation for architectural power measurement, and the use of a near-optimal search to tailor a processor design to a benchmark.

## 1 Introduction

Power limitations are increasingly nullifying some obvious superscalar advances. For example, duplicating commonly used functional units can enhance achievable parallelism [1]. Increasing duplication beyond current designs will increase power dissipation, yet current designs are already dissipating record amounts of power. Witness the 66MHz Intel Pentium that dissipates 16 watts or the 200MHz DEC Alpha AXP 21064 that dissipates 30 watts of power. Power is also directly affected by cycle time and improvements in cycle time are likely to complicate the current situation. Future increases in power dissipation may require expensive cooling and packaging techniques that significantly increase the system cost, pricing levels of performance out of the reach of all but supercomputer markets.

One source of the problem may be that architectural decisions are made largely without power usage information. Until now, obtaining such information before implementation has been extremely difficult. In the absence of this, the main emphasis has been on performance. Although high-performance designs may exist that dissipate low power, the only method for finding these designs has been through trial-and-error.

This paper presents a systematic technique to find low-power, high-performance superscalar processors tailored to specific user applications. Power in CMOS is composed of static and dynamic components. A novel approach is developed that separates the architectural contribution of dynamic power from the technology contribution. The architectural contribution is obtained via trace-driven simulation of SPEC92 benchmarks. The technology contribution is from a model based on estimates of actual designs. In reality, the constraints of circuit timing and limited real estate also impact power. Estimates of these are also included in the cost function. The function is optimized using a near-optimal search algorithm, to synthesize processor designs.

Duplication of integer functional units has been used in the Motorola 88110, the Intel Pentium and the Motorola/IBM PowerPC 604, among others [2],[3],[4]. The tradeoff between duplication and power has not been studied in detail. This paper uses the power model to address this question. Example superscalar designs are presented that achieve high parallelism

by duplicating functional units while dissipating in some cases less than 25% additional power over superscalars that have little duplicated hardware. The techniques used to select these designs and the estimates of power and area are presented in the next section. Comments, conclusions and future work close out the paper.

# 2 Methods and Models

The processor model for this study is a superscalar engine with full-Tomasulo scheduling and pipelined functional units. To achieve high parallelism, integer and floating-point functional units are duplicated and the functional unit latencies are optimized. This paper focuses on power-centric design of the processor's execution unit and its pool of functional units. For the Alpha 21064, this unit comprises slightly over half of the chip area (from micrographs presented in [5]).

The types of functional units are shown in Table 1. A 64-bit architecture is assumed. The integer class is composed of 64-bit integer ALU units (*IALU*), 64-bit shifter hardware (*Shift*) and branch hardware (*Branch*). The floating-point units are grouped into addition (*FPAdd*), multiplication (*FPMul*) and division (*FPDiv*). *FPDiv* is a pseudo-unit: division actually takes place in the multiplier using the quadratic convergence division method in an iterative, unpipelined fashion[1].

The data cache is accessed through three functional units: the *Load*, *Store* and *PMiss* units. PMiss is an abbreviation for *Pending Miss*. Any *Load* operation that causes a cache miss is automatically coupled with a dynamically created *PMiss* operation. These operations fetch the missing cache block independently from other cache accesses. Once a *PMiss* operation completes, its associated *Load* operation is allowed to execute. This scheme incorporates the lockup-free cache concepts presented by Kroft into a superscalar framework [8].

Example execution units are shown in Figure 1. In part (a) of the figure, an execution unit with no duplication is shown. This is the base design. Optimization for integer performance may result in design (b). Here the integer and the Load units have been duplicated. This allows parallel execution of independent integer instructions. A similar optimization for floating-point hardware may result in design (c).

[1] This algorithm can achieve the precision required by the IEEE standard at reasonable cost and speed [6] and was implemented in the RS/6000 [7]

Table 1: Functional unit types.

| Class | Unit | Description |
|---|---|---|
| Integer | IALU | Integer arithmetic, logicals |
| | Shift | Bit manipulation, shifting |
| | Branch | Branch prediction; fault recovery |
| Floating-Point | FPAdd | Floating-point addition |
| | FPMul | Integer,floating-point multiply |
| | FPDiv | Integer, floating-point divide |
| Data-Cache | Load | Data cache read |
| | Store | Store-buffer/data-cache write |
| | PMiss | Miss repair unit (lockup-free) |

## 2.1 An Architectural Power Model

Power dissipation in CMOS can be divided into a static and dynamic component. The static component is proportional to the product of gate leakage current (a function of the number of devices) times the supply voltage. This component is highly technology dependent. The dynamic component can be separated into architectural and technology components. To show this, assume a unit is pipelined into $N$ stages, labeled $S_1, S_2, \ldots, S_N$. Let $E_{S_i}$ be the *energy* consumed when stage $S_i$ performs work[2]. The power dissipated for one instruction is:

$$P = \frac{1}{T}\left(E_{S_1} + E_{S_2} + \cdots + E_{S_N}\right) \qquad (1)$$

where $T$ is the time it takes to execute the instruction (here $T = N$). Now consider a program fragment. Let $U_{S_i}$ be the total *usage* of stage $S_i$ during execution. The power dissipation now takes the form:

$$P = \frac{1}{T_{TOT}}\left(U_{S_1}E_{S_1} + U_{S_2}E_{S_2} + \cdots + U_{S_N}E_{S_N}\right), \qquad (2)$$

where $T_{TOT}$ is the total execution time for the program fragment. The stage energies, $E_{S_i}$, are *technology parameters*, whereas $T_{TOT}$ and the stage utilizations, $U_{S_i}$, are *architectural parameters*. This way, a simulation of the pipeline can measure the architectural parameters without any knowledge of the underlying technology.

An example helps illustrate the model. One popular myth about power usage is that pipelining can be ignored. The theory is that if any operation uses a unit, it must travel through all stages of the unit in turn, which means it consumes the same power (minus latching costs) as it would on an unpipelined unit. Figure 2 shows why this myth is false. (It has

[2] For now, we will assume this value is constant. Shortly, we will justify this approximation.

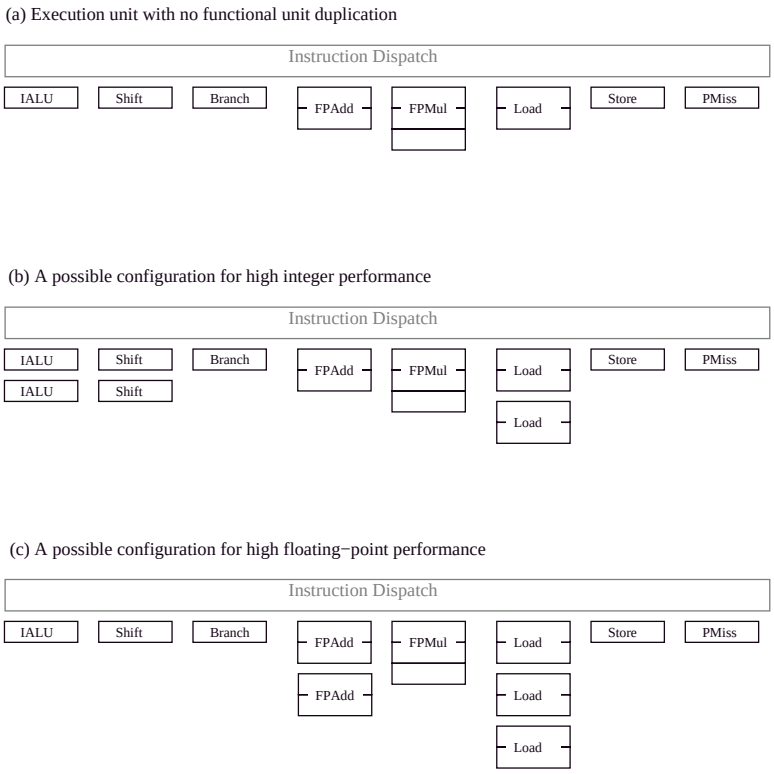


Figure 1: Example execution units.

also been disproved in [9].) Here three instructions are executed on a pipelined unit (left side of figure) and on an unpipelined unit (right side of figure). The corresponding power cost for each is shown below the figure. The unpipelined version uses 55% less power. The reason for this difference is the pipeline speedup effect, which is an architectural phenomenon.

The stage energy parameters $E_{S_i}$ are dependent on the logical inputs to the stages. They could also be derived via trace-driven simulation. However, for this study, a further approximation of stage energy is used. We assume that every device transitions when the unit is active, which is a worst-case assumption. We also assume that the worst case stage energy is related to the average energy by a constant.[3] An example model for the stage energies is presented below.

## 2.2 Simulation techniques

The set of benchmarks used in this paper is shown in Table 2. The benchmarks are compiled using GCC, which schedules instructions within basic blocks using a priority-based list scheduling algorithm. This shortens the critical dependence path of each block as much as possible, enhancing parallelism.

Architectural behavior is determined via trace-driven simulation. Traces are generated from benchmarks using the *Spike* tracing tool [10]. The simulator implements a dynamic instruction scheduling model, with the window for instruction scheduling moving between correctly predicted branches. Yeh's adaptive training branch algorithm is used to predict branch behavior, since it is currently one of the most accurate prediction schemes [11]. Since the benchmarks can generate extremely long traces, trace-sampling techniques are employed to reduce trace size and simulation time (see [12], [13] for details). Branch hardware and data cache simulation are done for the full traces, removing the possibility of sampling error for these units. The full trace is used to mark each incorrectly predicted branch in the sampled trace file. A similar approach is used to mark loads and stores that miss in the data cache.

The architectural component of dynamic power is measured during the simulation. In each cycle of the simulation, the usage of each pipeline stage is logged (i.e., $U_{S_i}$ is incremented if $S_i$ is busy). In addition, the simulator finds the total run time of the benchmark (i.e., $T_{TOT}$). This is later combined with the

[3] Even though the worst case is used for this study, it is not a *required* assumption of the model. A model based on average energy can be used for more accuracy. Our limited access to manufacturer implementations did not permit this.

Table 2: The benchmark set.

| Class | Benchmark | Description |
|---|---|---|
| Integer | compress | reduces the size of files |
| | eqntott | conversion from equation to truth table |
| | espresso | minimization of boolean functions |
| | gcc | GNU C compiler |
| | li | lisp interpreter |
| | sc | spreadsheet program |
| Floating-point | doduc | Monte Carlo simulation |
| | hydro2d | Solves Navier Stokes equations |
| | mdljdp2 | solves equations of motion |
| | ora | ray tracer through optical system |
| | tomcatv | vectorized mesh generation |
| | wave5 | solves Maxwell's equations |

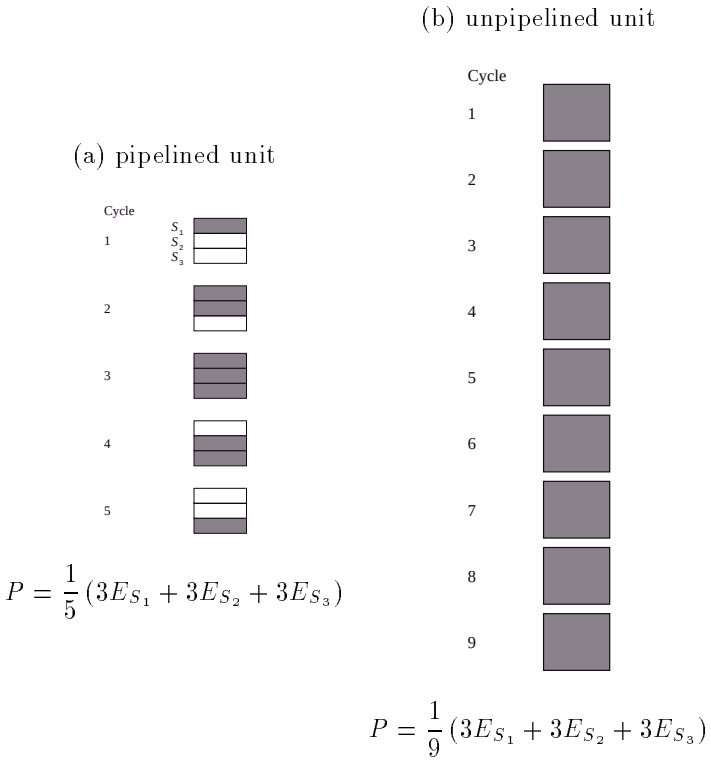


Figure 2: Why pipelining matters to power dissipation.

Part (a) shows a pipelined unit executing three instructions, (b) shows an unpipelined unit executing the same three instructions. Because of the effects of pipeline speedup on parallel stage usage, the power dissipated in (b) is $5/9 = 55\%$ smaller than (a).

technology parameters ($E_{S_i}$'s) to find the dynamic power component using Equation 2.

## 2.3 Finding power-efficient processors

One goal of this study is to determine designs that achieve high performance without excessive power or area usage. Each functional unit can be duplicated as many times as power and real estate limitations allow. This freedom of design results in an extremely large design space. Exhaustive search via simulation of this space is impractical. This problem lends itself to a near-optimal search, such as simulated annealing [14].

The following is the method used to guide the simulated annealing algorithm: Consider a processor design space composed of one or more of the functional units of Table 1, each having a latency ranging from 1 to $L_{\max}$. Let $P$ be some set of processors under consideration. A processor $p \in P$, has $n_j$ functional units of type $j$ and each of these functional units has a latency of $\ell_j$. A concise representation of processor $p$ is $p = \langle(\ell_0, n_0), (\ell_1, n_1), \ldots, (\ell_{k-1}, n_{k-1})\rangle$, for $k$ different types of functional units. At each step of the simulated annealing algorithm, the next design, $p_{i+1}$, is selected from the current design, $p_i$, using a restricted random selection procedure. The following procedure for determining a $p_{i+1}$ from a $p_i$ is used: $(i)$ $m$ functional units are selected at random from $p_i$, where $m$ is a random integer in the range $[1, 3]$, $(ii)$ the number of each of these functional units in $p_i$ is changed by a random integer in the range $[-3, 3]$. Any number greater than the issue rate or less than 1 is rejected. For units with several possible pipeline latencies, a slightly more restrictive procedure is used to randomly alter the latencies.

A superscalar without functional unit duplication

is used as the starting point for the search: $p_0 = \langle(\ell_{MIN}, 1)\rangle_k$, where $\ell_{MIN}$ are the minimal allowed latencies. The goal of the search algorithm is to adjust the parameters of $p_i = \langle(\ell_j, 1)\rangle_{1 \le j \le k}$ to maximize performance and yet remain within power and real estate budgets. A detailed description of this cost function is presented below.

## 2.4 Performance metrics

A performance metric is needed that takes into account both architectural performance and technological considerations. *Parallelism* or *instructions per cycle* (IPC) is often used for architectural performance. IPC is ultimately limited by the issue rate (a design feature) and inter-instruction dependencies (a benchmark characteristic).

IPC alone lacks technology considerations. For example, short latency functional units produce high IPC, since dependencies are resolved quicker using shorter latencies (shallow pipeline depths). However, lower degrees of pipelining may lengthen the execution unit's critical path. This has an impact on the total time to execute a program, but is not reflected by the IPC metric. The critical path that determines cycle time is typically through the first level of the memory hierarchy (e.g., the data cache). Shallow pipelines can shift this critical path into the execution unit. Since this paper concentrates on the execution unit, the aim is to optimize on the critical path within the execution unit. This reduces the impact of the execution unit's critical path on the external cycle time of the processor.

A metric that combines IPC and critical path delay is the *critical time per instruction* (CTPI). CTPI is the ratio of the critical path delay to the number of instructions per cycle. Optimizing the execution unit for low CTPI reduces the chance of affecting the processor's cycle time. For this reason, CTPI is used in the search algorithm's cost model.

## 2.5 Example technology cost model

It is exceedingly difficult to obtain accurate technology estimates of the state-of-the-art, since microprocessor manufacturers rarely release this information. In the absence of this, we have constructed what we believe to be a reasonably approximate model using published results. The model considers a processor implementation technology with a budget of 1.68 million transistors and a supply voltage of 3.3 volts. This is based on the reported figures in [5] for a 0.75$\mu$m three metal-layer CMOS process technology.
Although the first-level data cache is not included in the execution unit, its miss rate impacts the overall performance of the superscalar core. A 16KB, 2-way associative data cache is assumed. This design assumes a page size of 8K bytes so that cache data store indexing can occur in parallel with TLB access. Cache misses are handled by the hardware using a lockup-free mechanism [8]. The latency to repair a missing block from the L2 cache is assumed to be 10 cycles.

The specific cost model depends on **performance**, **real estate** and **power** estimates:

**Performance**: Performance is measured by CTPI, where low CTPI is desirable. CTPI is calculated from the number of instructions, the number of cycles for the execution of the program, and an estimate of the critical path. The deepest pipeline stage in the execution unit is used to find the critical path using a technique presented by the authors in [13]. The function units could not be partitioned such that the cycle time is *exactly* inversely proportional to the degree of pipelining. Instead the deepest pipeline stage for each degree of pipelining is determined from our own designs. The sum of the gate delays within this stage constitutes the cycle time.

**Real estate**: Transistor level analysis of published work provided the approximations for each functional unit type. This model is presented in Table 3. (Since the *FPMul* unit is used iteratively for division, the *FPDiv* unit does not consume any die space and is not mentioned in the table.)

Real estate goals are expressed as a budget:

$$(\text{area of } p_i) \le (\text{area budget})$$

The area budget is based on the 1.68 million transistor budget, which includes interconnection overhead. Approximating interconnection overhead as half of this figure, and assuming that the execution unit comprises half of the total die (as is the case with the Alpha 21064), the real estate budget is taken as 25% of 1.68 million, or 420,000.

**Power**: Only relative power increases are required for the cost model. Therefore, power is normalized. Exact gate leakage currents for the two technologies are unpublished, but dynamic power is taken to be approximately 10,000 times larger than static power (a typical ratio). Static power is estimated using the product of the total number of transistors (which is proportional to the leakage current) times the supply voltage. The trace-driven simulation model is used for dynamic power. As stated above, the worst-case estimates are used to calculate the stage energies. This component is weighted by the square of the supply voltage [17].

Table 3: Real estate usage by functional unit type.

| Functional unit | Allowed latencies | Number of transistors (by latency) 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| IALU | 1–1 | 5068 | – | – | – | – | – |
| Shift | 1–1 | 6272 | – | – | – | – | – |
| Branch | 1–1 | 8660 | – | – | – | – | – |
| FPAdd* | 1–5 | 18880 | 19192 | 19504 | 19504** | 19816 | – |
| FPMul* | 1–6 | 40292 | 41540 | 46196 | 42788 | 43796 | 46436 |
| Load | 1–4† | 4928 | 4928 | 4928 | 4928 | – | – |
| Store | 1–1 | 4928 | – | – | – | – | – |
| Pmiss | 10 | 46848‡ | 46848 | 46848 | 46848 | 46848 | 46848 |

*Sources: [15],[16],[6] along with our own implementations.
**No change is seen in the number of transistors from latency 3 to 4 since the placement of the latches results in fewer bits that need to be latched.
†Load is through the data cache, which is excluded from the execution unit. However, slight overhead is required for each load operation to latch the values. Multiple load units are implemented by interleaving the cache.
‡Value shown is extrapolated from [8].

The power consumed by a processor $p_i$ is constrained to a fractional increase in the power consumed by a processor without duplicated functional units (processor $p_0$):

$$(\text{power of } p_i) \leq K \times (\text{power of } p_0).$$

The overall goal is to minimize CTPI subject to constrained power and area budgets. An expression for the combined cost function is:

$$f(p_i) = \begin{cases} \text{CTPI}, & \text{if (area of } p_i) \leq \text{(area budget) \&} \\ & \text{(power of } p_i) \leq K \times \text{(power of } p_0) \\ \infty, & \text{otherwise.} \end{cases}$$

# 3 Experimental Results

This section presents examples of the method in action. The example cost model is used to optimize processors for increased performance via duplicated functional units. Power is limited to a fractional increase over the base case (no duplicated hardware). The power budget factor, $K$, is selected such that the limit of power usage is quite restrictive. Assuming that the execution unit occupies 50% of the chip area and that the power usage is uniformly distributed across the chip, a power budget of $K = 1.5$ translates into a 25% increase in power overall. This sometimes restricts performance improvement. A second budget of a 50% increase in power ($K = 2.0$) is also investigated.

Figure 3 illustrates the evolution of the cost function for *li*. As may be seen, an immediate attempt is

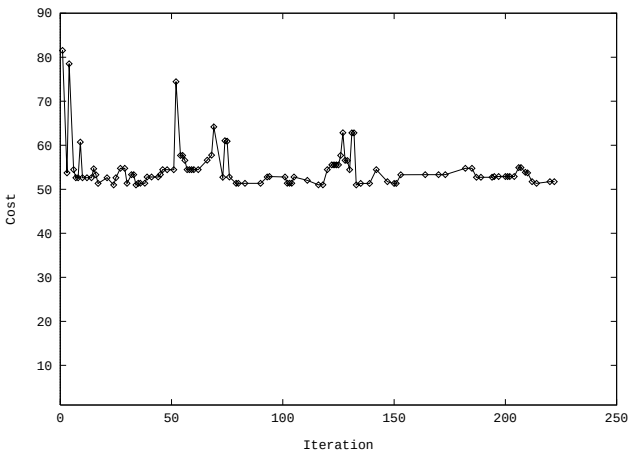


Figure 3: The cost function for *li* (infinities excluded from plot).

Table 4: Performance of initial designs.

| Class | Benchmark | IPC | Power Index |
|---|---|---|---|
| Integer | compress | 1.40 | 19.83 |
| | eqntott | 1.38 | 20.80 |
| | espresso | 1.41 | 21.03 |
| | gcc | 1.27 | 20.28 |
| | li | 1.40 | 21.10 |
| | sc | 1.36 | 25.41 |
| Floating point | doduc | 1.80 | 34.82 |
| | hydro2d | 1.85 | 28.51 |
| | mdljdp2 | 2.05 | 34.58 |
| | ora | 1.82 | 26.42 |
| | tomcatv | 2.13 | 47.23 |
| | wave5 | 1.79 | 28.09 |

made to reduce the cost from that of the initial design, $p_0$. Although the new cost is better than the original, the search continues for a more global minimum. The search is initially liberal in its design selections but settles into a region of the design space after the 150th iteration.

## 3.1 Performance of initial designs

Table 4 shows the performance of the initial designs (no duplicated functional units). Also presented is the normalized power index from the power estimators. The power index for the floating-point benchmarks is consistently higher than the integer benchmarks. Floating-point units burn higher amounts of power than integer units, due to a higher number of transistors per unit. Note the strong correlation between high IPC and high power usage: more instructions executing in parallel implies more functional units active. This correlation between power and instruction-level parallelism implies high-performance superscalars are high power designs. Our goal is to shift performance gains to less power-intensive functional units by duplicating those units.

## 3.2 Optimized designs

The optimized designs for each benchmark are presented below. The IPC and power index values[4] of the designs are presented graphically in Figures 4 and 5. Comparison of Figure 4 and Figure 5 shows the a general correlation between IPC and the power index. As the power budget is increased, IPC (parallelism) also increases. This is only true on a per-benchmark basis, and not true across benchmarks. Although for 125% of initial power, the IPC for *li* is greater than that for *doduc*, the power index for *doduc* exceeds that for *li*. This is because the floating-point intensive operations found in *doduc* use more power than integer intensive operations in *li*.

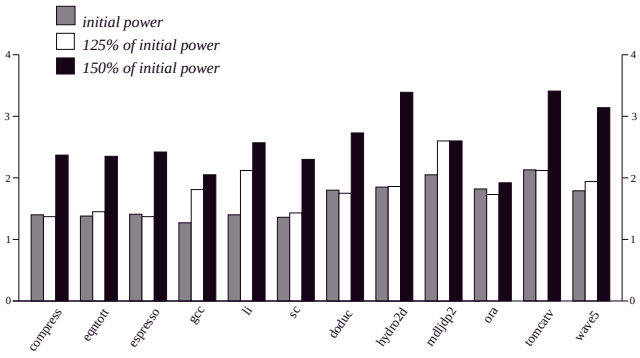


Figure 4: IPC for the benchmarks.

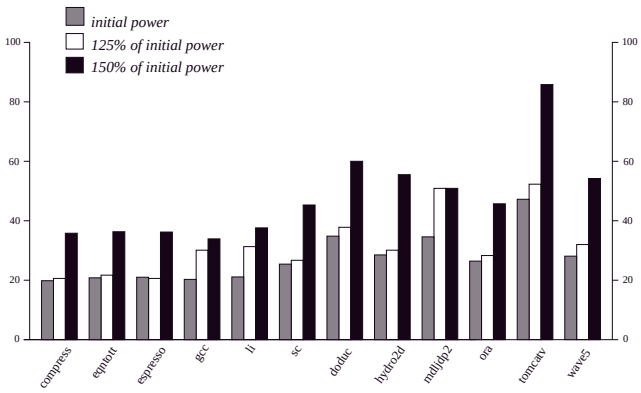


Figure 5: Power usage for the benchmarks.

Table 5 present the optimized designs for power budgets of 125% (part (a)) and 150% (part (b)).

### 125% power budget designs

The integer-intensive benchmarks in Table 5(a) achieve significant performance increases over the initial designs in Table 4. For example, the IPC for *gcc* improves from 1.27 initially to 1.81, and *li* increases from 1.40 to 2.12. These increases are achieved primarily by replicating the $IALU$ units. The floating-point benchmarks do not have use for more than one $IALU$. The exception is *mdljdp2*, which uses four, due to a high number of address (array indexing) calculations.

[4] IPC is used here to measure external performance, assuming minimization of the critical path did not affect the external cycle time (see Section 2.4).

Table 5: The power/area-efficient processor designs

(a) power budget 125%

| Benchmark | IPC | Power Index | IALU $n$ | IALU $\ell$ | Shift $n$ | Shift $\ell$ | Branch $n$ | Branch $\ell$ | FPAdd $n$ | FPAdd $\ell$ | FPMul $n$ | FPMul $\ell$ | Load $n$ | Load $\ell$ | Store $n$ | Store $\ell$ | PMiss $n$ | PMiss $\ell$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **compress** | 1.37 | 20.6 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 5 | 1 | 6 | 2 | 1 | 1 | 1 | 1 | 10 |
| **eqntott** | 1.45 | 21.7 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 5 | 1 | 6 | 2 | 1 | 1 | 1 | 1 | 10 |
| **espresso** | 1.37 | 20.6 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 5 | 1 | 6 | 2 | 1 | 1 | 1 | 1 | 10 |
| **gcc** | 1.81 | 30.1 | 2 | 1 | 2 | 1 | 2 | 1 | 1 | 5 | 1 | 6 | 3 | 3 | 1 | 1 | 2 | 10 |
| **li** | 2.12 | 31.3 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | 1 | 6 | 4 | 3 | 2 | 1 | 2 | 10 |
| **sc** | 1.43 | 26.7 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 5 | 1 | 6 | 2 | 2 | 1 | 1 | 1 | 10 |
| **doduc** | 1.75 | 37.8 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 5 | 2 | 6 | 1 | 1 | 2 | 1 | 1 | 10 |
| **hydro2d** | 1.86 | 30.1 | 1 | 1 | 1 | 1 | 3 | 1 | 2 | 5 | 1 | 6 | 2 | 1 | 2 | 1 | 1 | 10 |
| **mdljdp2** | 2.60 | 50.9 | 4 | 1 | 1 | 1 | 3 | 1 | 1 | 5 | 3 | 6 | 3 | 1 | 3 | 1 | 1 | 10 |
| **ora** | 1.73 | 28.3 | 1 | 1 | 1 | 1 | 3 | 1 | 2 | 5 | 1 | 6 | 2 | 1 | 2 | 1 | 1 | 10 |
| **tomcatv** | 2.12 | 52.3 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 5 | 2 | 6 | 2 | 1 | 1 | 1 | 1 | 10 |
| **wave5** | 1.94 | 32.0 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 5 | 2 | 6 | 3 | 1 | 2 | 1 | 1 | 10 |

(b) power budget 150%

| Benchmark | IPC | Power Index | IALU $n$ | IALU $\ell$ | Shift $n$ | Shift $\ell$ | Branch $n$ | Branch $\ell$ | FPAdd $n$ | FPAdd $\ell$ | FPMul $n$ | FPMul $\ell$ | Load $n$ | Load $\ell$ | Store $n$ | Store $\ell$ | PMiss $n$ | PMiss $ell$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **compress** | 2.37 | 35.8 | 3 | 1 | 2 | 1 | 1 | 1 | 1 | 5 | 1 | 6 | 1 | 1 | 3 | 1 | 1 | 10 |
| **eqntott** | 2.35 | 36.3 | 3 | 1 | 1 | 1 | 2 | 1 | 1 | 5 | 1 | 6 | 3 | 2 | 1 | 1 | 3 | 10 |
| **espresso** | 2.42 | 36.2 | 4 | 1 | 2 | 1 | 2 | 1 | 1 | 5 | 1 | 6 | 3 | 1 | 2 | 1 | 2 | 10 |
| **gcc** | 2.05 | 33.9 | 3 | 1 | 2 | 1 | 2 | 1 | 1 | 5 | 1 | 6 | 2 | 1 | 2 | 1 | 2 | 10 |
| **li** | 2.57 | 37.6 | 4 | 1 | 1 | 1 | 3 | 1 | 1 | 5 | 1 | 6 | 3 | 1 | 2 | 1 | 2 | 10 |
| **sc** | 2.30 | 45.3 | 3 | 1 | 2 | 1 | 3 | 1 | 1 | 5 | 1 | 6 | 3 | 1 | 1 | 1 | 1 | 10 |
| **doduc** | 2.73 | 60.0 | 4 | 1 | 1 | 1 | 2 | 1 | 1 | 5 | 3 | 6 | 3 | 1 | 1 | 1 | 3 | 10 |
| **hydro2d** | 3.39 | 55.5 | 4 | 1 | 4 | 1 | 2 | 1 | 2 | 5 | 3 | 6 | 3 | 1 | 1 | 1 | 2 | 10 |
| **mdljdp2** | 2.60 | 50.9 | 4 | 1 | 1 | 1 | 3 | 1 | 1 | 5 | 3 | 6 | 3 | 1 | 3 | 1 | 1 | 10 |
| **ora** | 1.92 | 45.7 | 3 | 1 | 2 | 1 | 2 | 1 | 2 | 5 | 1 | 6 | 3 | 1 | 1 | 1 | 1 | 10 |
| **tomcatv** | 3.41 | 85.8 | 2 | 1 | 2 | 1 | 2 | 1 | 1 | 5 | 2 | 6 | 3 | 1 | 1 | 1 | 1 | 10 |
| **wave5** | 3.14 | 54.2 | 4 | 1 | 3 | 1 | 2 | 1 | 2 | 5 | 2 | 6 | 4 | 2 | 2 | 1 | 2 | 10 |

There is little performance increase for the floating-point benchmarks, and in some cases the designs actually achieve a slight performance decrease. The 125% budget appears to be too restrictive for these benchmarks, preventing simulated annealing from improving the designs considerably. This is also the reason that floating-point unit duplication does not necessarily result in higher performance. This is not the case when a budget of 150% is assumed (see below).

The memory access units are optimized for multiple copies across all benchmarks. Load ports are especially important for most benchmarks, with four load ports required for *li*. Note that multiple load ports translate into high degrees of cache interleaving. With the exception of *gcc* and *li*, no more than one pending miss is needed.

In general, the parallelism available from a four instructions per cycle lookahead is wasted for the floating-point benchmarks under the 125% budget.

### 150% power budget designs

The designs constrained by a 150% power budget are presented in Table 5(b). Increasing the power budget to twice the initial design has a dramatic effect on the achieved parallelism of the benchmarks. The integer benchmarks achieve parallelism values as high as 2.57 instructions per cycle (*li*). The floating-point benchmarks now achieve values significantly better than those of the Table 4 initial designs. *Tomcatv*, for example, achieves an IPC of 3.41 versus 2.13. Functional unit duplication favors the integer units, as before. *Espresso* now uses four *IALU* units, instead of the one used for the 125% power budget. In general, the 150%-budget designs are elite,

high-performance superscalars. The *hydro2d* design achieves an IPC = 3.39, for example. In general, all IPC values are greater than 2.0, with the exception of *ora* (1.92). The most-popular unit to duplicate remains the $IALU$ unit, with a median value of three $IALU$ units. The $IALU$ is one of the lowest-power units and at the same time one of the most-needed units. Other integer units are duplicated as well: *hydro2d* selects four *Shift* units and *wave5*, three[5].

The memory access units follow the general trend observed for the 125% power budget. Load ports are very important. Also, two of the floating-point benchmarks and one of the integer benchmarks require parallel data cache writes (three for *compress* and *mdljdp2*). In some cases, three pending miss units are required.

When combined, these results suggest that high performance can be achieved by technology-based allocation of units that deliver high performance without high power usage.

# 4 Conclusion

The number and importance of technology considerations has increased dramatically in recent years with advances in high-performance processor designs. The techniques presented in this paper make the connection between these technology considerations and the architecture level. The major contributions of this paper are the separation of architectural and technology components of dynamic power, the use of trace-driven simulation for architectural power measurement, and the use of a near-optimal search to tailor a processor to a benchmark. Although based on published design information, the example technology model has some flaws. It was presented in order to demonstrate the technique. Industry practitioners can readily develop a more accurate technology model for their internal use.

The model's predictions do match conventional wisdom: integer ALU's are best to duplicate, floating-point hardware is expensive and power-hungry, etc. With caveats in mind, several insights can be drawn from the results:

- Integer units (IAlu and Shift) achieve the highest performance per power increase,
- For higher performance designs, up to three pending misses may be required (note that this result is only valid for a miss repair penalty of 10),
- Deeply pipelined floating-point units are favored for their cycle time advantage,
- Provisions for multiple, parallel data cache reads are required,
- Up to three speculative branches active at once are needed in some cases (the PowerPC 604 currently supports two [4]),

In general, significant increases in floating-point performance will require much higher power budgets than equivalent increases in integer performance. Significant performance via duplication cannot be contemplated without taking this effect into account. The techniques in this paper find and duplicate units that produce the most performance for the least amount of power dissipation.

[5] Shifts are used for array index multiplication after the compiler applies strength reduction.

## References


[1] T. M. Conte, "Architectural resource requirements of contemporary benchmarks: A wish list," in *Proc. 26th Hawaii Int'l. Conf. on System Sciences*, vol. 1, (Maui, HI), pp. 517–529, Jan. 1993.

[2] D. R. Ditzel and H. R. McLellan, "Branch folding in the CRISP microprocessor: Reducing branch delay to zero," in *Proc. 14th Ann. International Symposium Computer Architecture*, (Pittsburgh, PA), pp. 2–9, June 1987.

[3] D. Alpert and D. Avnon, "Architecture of the Pentium microprocessor," *IEEE Micro*, vol. 13, pp. 11–21, June 1993.

[4] S. P. Song and M. Denman, "The PowerPC 604 RISC microprocessor," tech. rep., Somerset Design Center, Austin, TX, Apr. 1994.

[5] E. McLellan, "The Alpha AXP architecture and the 21064 processor," *IEEE Micro*, vol. 13, pp. 36–47, June 1993.

[6] I. Koren, *Computer arithmetic algorithms*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[7] P. W. Markstein, "Computation of elementary functions on the IBM RISC system/6000 processor," *IBM J. Research and Development*, vol. 34, pp. 111–119, Jan. 1990.

[8] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proc. 8th Ann. Int'l. Symp. Computer Architecture*, pp. 81–87, May 1981.

[9] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.

[10] M. L. Golden, "Issues in trace collection through program instrumentation," Master's thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, Illinois, 1991.

[11] T. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proc. 24th Ann. International Symposium on Microarchitecture*, (Albuquerque, NM), pp. 51–61, Nov. 1991.

[12] T. M. Conte, *Systematic computer architecture prototyping*. PhD thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois, 1992.

[13] T. M. Conte and W. Mangione-Smith, "Determining cost-effective multiple issue processor designs," in *Proc. 1993 Int'l. Conf. on Computer Design*, (Cambridge, MA), Oct. 1993.

[14] S. Kirkpatric, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[15] D. W. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "The IBM system/360 model 91: Floating point execution unit," *IBM J. Research and Development*, vol. 11, pp. 25–33, Jan. 1967.

[16] T. Asprey, G. S. Averill, E. DeLano, R. Mason, B. Weiner, and J. Yetter, "Performance features of the PA7100 microprocessor," *IEEE Micro*, vol. 13, pp. 22–35, June 1993.

[17] M. Annaratone, *Digital CMOS circuit design*. Boston, MA: Kluwer Academic Publishers, 1986.