System-Level Power Consumption Modeling and Tradeoff Analysis Techniques for Superscalar Processor Design

Thomas M. Conte *Member, IEEE*, and
Kishore N. Menezes *Student Member, IEEE*, and
Sumedh W. Sathaye *Student Member, IEEE*

Manuscript received _____.

Affiliation: The authors are with the Department of Electrical and Computer Engineering, P. O. Box 7911, North Carolina State University, Raleigh, NC 27695–7911. E-mail: conte@eos.ncsu.edu .

This is a revised and expanded version of the paper presented by the authors at the 28^{th} Hawaii International Conference on System Sciences (Jan. 1994), Maui, HI.

Key-Phrases

SUPERSCALAR

POWER DISSIPATION

INSTRUCTION-LEVEL PARALLELISM

NEAR-OPTIMAL SEARCH

HIGH-LEVEL SYNTHESIS

Abstract

High-level decisions in high-performance processors are often decoupled from their ultimate impact on power usage. For example, superscalar hardware and high degrees of pipelining are excellent sources for high parallelism. They often result in higher power usage. This problem is further complicated by the usage patterns of each unit in the processor. The usage patterns are determined by the programs the system executes, and ultimately by the applications the processor is targeted towards.

This paper presents systematic techniques to find low-power, high-performance superscalar processors tailored to specific user applications. 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: decreasing power consumption in a superscalar processor without greatly impacting performance. Results are presented from runs using simulated annealing to reduce power consumption subject to performance reduction bounds.

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.

I. Introduction

All recent processor offerings are superscalar designs. These processors use duplicated, independent functional units to execute instructions in parallel. The ability to execute in parallel is limited by the flow of information between instructions, since some instructions depend on results calculated earlier in the program. Superscalar processor organizations use hardware techniques such as the Tomasulo algorithm [1] to detect parallelism and execute code correctly. Empirical results suggest as much as a five times speed improvement when instruction-level parallelism is exploited [2]. Current designs seek parallelism by examining and issuing four to six instructions per cycle, with higher rates expected [3],[4],[5],[6]. Successful use of these high issue rates requires careful tuning of the microarchitecture. There is a wealth of technological alternatives for this task. These include branch handling strategies [7], functional unit duplication [2], and instruction fetch, issue, completion and retirement policies [8]. The deciding factor between the

various techniques is a function of the performance each adds, versus the cost each incurs. Unfortunately, this tradeoff analysis rarely takes power consumption into account. Consequently, current superscalar processors consume 30-40 watts of power.

The organization of a high-performance microprocessor is determined using results from behavioral simulations. Performance is measured as the number of cycles per instruction or the overall run time for a set of test programs. Power consumption is not considered until much later in the design process. As such, it is the responsibility of the circuit designers rather than the architects. However, parallelism and pipelining have direct impact on processor designs. Highly-parallel processors consume more power per cycle than non-parallel hardware. Deeply-pipelined functional units consume more power, since the power is consumed over a shorter period of time. This suggests tradeoffs between power consumption and processor organization that defy simple, rule-of-thumb approaches.

This research develops a system-level, behavioral model of power consumption for designing low-power, high-performance superscalars. This model is a separable cost function that can be used to optimize superscalars. The cost is separated into organizational and technological components. The organizational component is measurable from a behavioral-level simulation of the type used for high-level design. The technological component depends on the implementation technology. The components can be combined after simulation to estimate power dissipation. A near-optimal search algorithm is employed to reduce the power consumption of superscalar processor designs without high sacrifices in performance. The combined cost function and near-optimal search method is suitable for tradeoff analysis of processor organizations. The method introduces power considerations into the organizational design process, reducing overall power consumption through organizational changes.

II. 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 varied. 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 [4]. The

execution unit has 9 functional units, the types of functional units are shown in Table I. A 64-bit word size 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]. All units are designed using static CMOS with input buffering.

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 unit incorporates the lockup-free cache design presented by Kroft [11].

TABLE I

FUNCTIONAL UNIT TYPES.

Class	Model index	Functional Unit	Description
Integer	0	IALU	Integer arithmetic, logicals
	1	Shift	Bit field manipulation, shifting
	2	Branch	Branch prediction and fault recovery
Floating-	3	FPAdd	Floating-point addition
Point	4	FPMul	Integer and floating-point multiplication
	5	FPDiv	Integer and floating-point division
Data-	6	Load	Data cache read
Cache	7	Store	Store-buffer based data-cache write
	8	PMiss	Missed block repair unit (lockup-free cache)

Consider a processor design space composed of one or more of the functional units of Table I, each having a latency ranging from 1 to $L_{\max}$. Let M be the set of processors under consideration. A processor $m \in M$, has n_j functional units of type j and each of

[1]This algorithm can achieve the precision required by the IEEE standard at reasonable cost and speed [9] and was implemented in the RS/6000 [10]

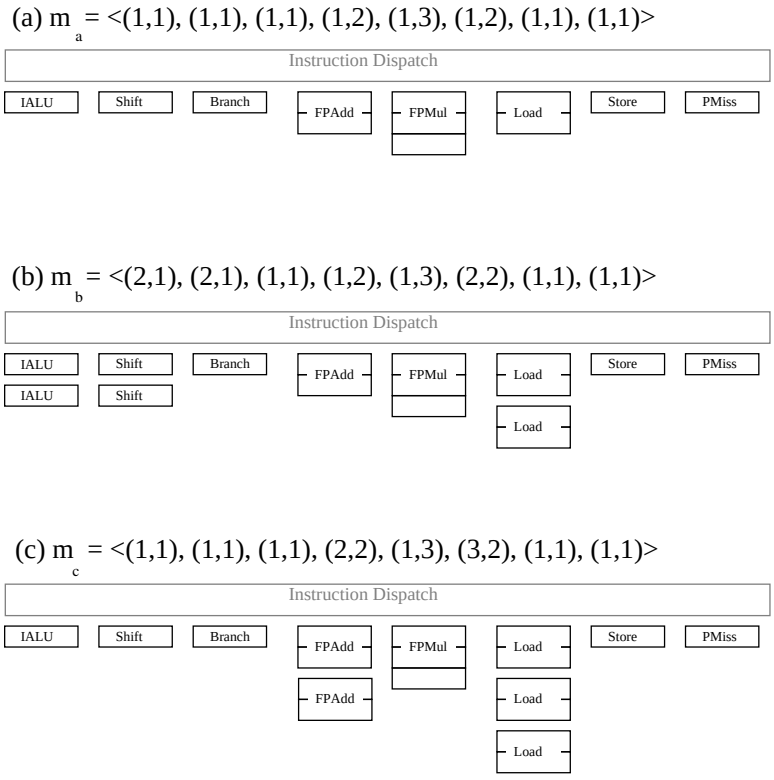

Fig. 1. Example processor designs.

these functional units has a latency of ℓ_j, such that,

$$m = \langle (n_0, \ell_0), (n_1, \ell_1), \ldots, (n_{s-1}, \ell_{s-1}) \rangle, \tag{1}$$

for s different types of functional units. The block diagrams of the execution units for three example processor designs are depicted in Figure 1. In part (a) of the figure, m_a has an execution unit with no duplication. This design is limited to parallelism between heterogeneous instruction types. Optimization for integer performance may result in m_b (Figure 1(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 m_c (Figure 1(c)).

A. A System-Level Power Model

Excessive power dissipation is known to cause serious packaging and thermal problems. Some instances are the 30 watts dissipated by the 200MHz DEC Alpha AXP 21064 and the 16 watts dissipated by the 66MHz Intel Pentium. As clock rates increase, this aspect

of design gains equal importance as the performance and die space.

Power dissipation in static CMOS can be divided into a static and dynamic component. Static power dissipation is due to the reverse bias leakage current between diffusion regions and the substrate during steady state. This component is highly technology dependent. The static power dissipation, P_{static}, for a particular functional unit is estimated by,

$$P_{\text{static}} \doteq S_T \times I_{\text{leakage}} \times V_{DD}, \tag{2}$$

where S_T is the size of the functional unit in transistors, I_{leakage} is the leakage current, and V_{DD} is the supply voltage.

Dynamic power dissipation can be separated into system-level and technology components. To show this, assume a unit is pipelined into N stages, labeled $S_1, S_2, \ldots, S_N$. Let $\bar{E}_{S_i}$ be the *average energy* consumed when stage S_i performs work. The average dynamic power dissipated to execute a single instruction is,

$$P_{\text{dyn}} = \frac{1}{T}\left(\bar{E}_{S_1} + \bar{E}_{S_2} + \cdots + \bar{E}_{S_N}\right), \tag{3}$$

where T is the time it takes to execute the instruction (here $T = N$).

Now consider a program fragment containing multiple instructions. Let U_{S_i} be the total *usage* of pipeline stage S_i during execution. The power dissipation now takes the form:

$$P_{\text{dyn}} = \frac{1}{T_{TOT}}\left(U_{S_1}\bar{E}_{S_1} + U_{S_2}\bar{E}_{S_2} + \cdots + U_{S_N}\bar{E}_{S_N}\right), \tag{4}$$

where T_{TOT} is the total execution time for the program fragment. The stage energies, $\bar{E}_{S_i}$, are *technology parameters*, whereas T_{TOT} and the stage utilizations, U_{S_i}, are *system-level parameters*. A behavioral simulation of the pipeline can be used to obtain the system-level utilizations without knowledge of the underlying technology. This power model is similar to the instruction level power model introduced in [12].

An example helps illustrate the model. System designers often assume that pipelining does not affect power dissipation. The theory is that if any instruction uses a functional unit, it must travel through all stages of the unit in turn, which implies it consumes the same power as it would on an unpipelined unit (neglecting latching costs). Figure 2 shows why this assumption is false. Here three instructions are executed on a pipelined unit (Figure 2(a)) and on an unpipelined unit (Figure 2(b)). The corresponding power cost

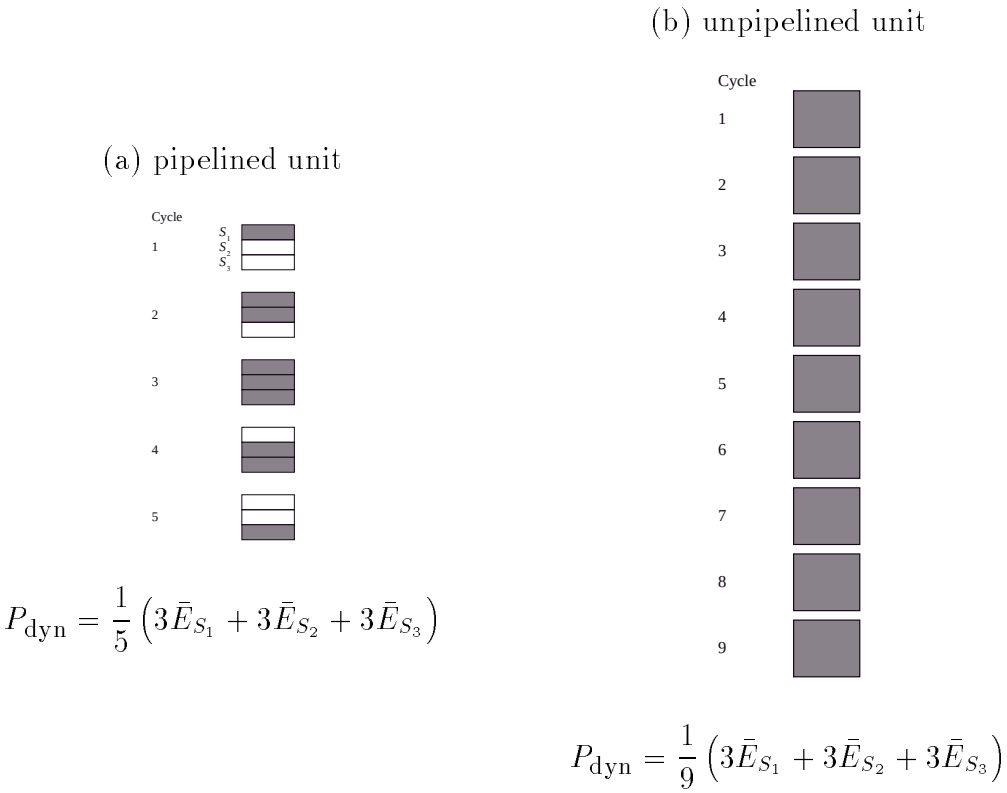

Fig. 2. Example demonstrating the reason that pipelining is significant to power dissipation. Part (a) depicts a pipelined unit executing three instructions (neglecting latches), (b) depicts an unpipelined unit executing the same three instructions. Because of the effects of pipeline speedup on parallel stage usage, the power dissipated in (b) is $5/9 = 55\%$ of (a).

for each is shown below the figure. The unpipelined version uses 55% of the power of the pipelined version. The reason for this difference is the pipeline speedup effect, which is an architectural phenomenon. The assumption that pipelining does not matter has also been persuasively disproved in [13].

The total dynamic power consumption can be calculated from the power consumptions of each unit. Let S_{ijk} be the ith pipeline stage in jth copy of functional unit type k. The total dynamic power consumption, P_{TOT}, is then,

$$P_{\text{dyn}} = \frac{1}{T_{TOT}} \sum_{k=0}^{s-1} \sum_{j=0}^{n_k-1} \sum_{i=0}^{\ell_i-1} U_{S_{ijk}} \bar{E}_{S_{ijk}}. \tag{5}$$

The values of the stage energy parameters, $\bar{E}_{S_{ijk}}$, are dependent on the logical inputs to the stages. This study uses an approximate model of $\bar{E}_{S_{ijk}}$ that assumes each device in stage S_i

transitions when the stage is active. This tends to over-estimate power consumption. The goal of the technique presented here is to achieve low-power, high-performance designs. Over-estimation of $\bar{E}_{S_i}$ is consistent with this goal. Further details concerning this model are presented below.

B. Simulation techniques

System-level design employs trace-driven behavioral simulation, where the traces are taken from a set of industry-standard benchmark programs. Members of the SPEC92 workstation benchmarks [14] are used here, summarized in Table II. The benchmarks are compiled using the public-domain GNU C compiler, which implements an aggressive set of code-improving optimizations, including a priority-based list scheduling algorithm [15]. This shortens the critical dependence path of instruction sequences as much as possible, enhancing parallelism between instructions and resulting in higher superscalar processor performance. The traces of the benchmarks are generated from benchmarks using the *Spike* tracing tool [16].

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 [17]. Since the benchmarks can generate extremely long traces, trace-sampling techniques are employed to reduce trace size and simulation time (see [18], [19] for details). Only the pipeline state is sampled, the entire memory system including branch hardware and caches are simulated using the full trace. This results in a relative error of no more than $\pm 3\%$ for the processor performance metrics. During simulation, the values of pipeline stage usage are calculated (U_{S_i} is updated if S_i is busy). The simulator also estimates the total run time of the benchmark (T_{TOT}). After simulation, this information is combined with the technology parameters ($\bar{E}_{S_i}$'s) to find the dynamic power component using Equation 5. The total power is then estimated by summing the dynamic component with the static component (Equation 2).

TABLE II

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

C. Tradeoff analysis using near-optimal search

One goal of this study is to determine designs that achieve low power without sacrificing superscalar performance. To achieve this, a high-performance processor with duplicated functional units is used as the starting point. Each functional unit can be duplicated as many times as power constraints allow. This freedom of design results in an extremely large design space. Exhaustive search via behavioral simulation of this space is computationally impractical. This problem lends itself to application of a near-optimal search algorithm. A variant of simulated annealing is employed here for this task [20].

The following is the method used to guide the simulated annealing algorithm: At each step of the algorithm, the next processor design, m_{i+1}, is derived from the current design, m_i, using a restricted random selection procedure. The random selection procedure is: (1) select l functional units at random from m_i, where l is a random integer in the range $[1, 3]$,

(2) the number of each of these functional units in m_i is changed by a random integer in the range $[-3, 3]$. Any number greater than the issue rate (four instructions per cycle) 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.

The initial design used as the starting point for the search is:

$$m_0 = \langle(\mathrm{IR}, \ell_0), (\mathrm{IR}, \ell_1), (3, \ell_2), (\mathrm{IR}, \ell_3), (\mathrm{IR}, \ell_4), (1, \ell_5), (\mathrm{IR}, \ell_6), (\mathrm{IR}, \ell_7), (\mathrm{IR}, \ell_8)\rangle. \quad (6)$$

where ℓ_i is the minimal allowed latency for functional unit of type i, and IR is the issue rate. All units are replicated to a degree of IR, with the exception of three branch units (unit type 2) and one FPDiv pseudo unit (unit type 5). The goal of the search algorithm is to adjust the design parameters of m_i to minimize power and yet achieve superscalar performance comparable to m_0. A further description of the cost function is presented below.

D. Performance metrics

A performance metric is used that takes into account both performance due to processor organization and due to 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. Hence, tradeoff analysis employing only IPC would result in a sub-optimal design.

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 study concentrates on the superscalar execution unit, the aim is to optimize the critical path within the pipelines of the functional units. 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.

E. Example technology cost model

The example technology cost model considers a processor implementation technology with a budget of 1.7 million transistors and a supply voltage of 3.3 volts. This is based on the reported figures in [4] for a 0.75μm 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 [11]. The latency to repair a missing block from the L2 cache is assumed to be 10 cycles.

The specific cost model depends on CTPI and power consumption estimates. 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 employing a technique presented in [19]. It is rarely true that the functional units can be pipelined 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. The sum of the device propagation delays within this stage constitutes the cycle time.

The CTPI increase of processor m_i, CTPI(m_i), is constrained to a fractional increase over CTPI(m_0):

$$\text{CTPI}(m_i) \leq K \times \text{CTPI}(m_0), \tag{7}$$

where K is the CTPI budget.

Transistor level analysis of published work provided the approximations for each functional unit type. This model is presented in Table III. (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.)

TABLE III

SUMMARY OF TECHNOLOGY MODEL BY FUNCTIONAL UNIT TYPE.

Functional unit	Allowed latencies	Number of transistors (by pipeline 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: [21],[22],[9] 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 [11].

Only relative power dissipation increases are required for the cost model. Therefore, the power estimate is normalized to remove any multiplicative error in the model. The coefficients are adjusted such that dynamic power is 10,000 times larger than static power for a single device (a typical ratio). Static power is estimated using Equation 2. Equation 5 is used to estimate dynamic power. Stage energies are calculated using the functional unit designs of Table III.

The overall goal is to minimize power subject to constrained performance degradation. An expression for the combined cost function is,

$$f(m_i) = \begin{cases} \text{power of } m_i, & \text{if CTPI}(m_i) \leq K \times \text{CTPI}(m_0), \\ \infty, & \text{otherwise.} \end{cases} \tag{8}$$

III. Experimental Results

This section presents example results of the system-level power dissipation model and tradeoff analysis method. The initial design, m_0, is selected using Equation 6 with the issue rate equal to four instructions per cycle (IR = 4). Figure 3 illustrates the evolution of the cost function during a near-optimal search for the *espresso* benchmark. As may be

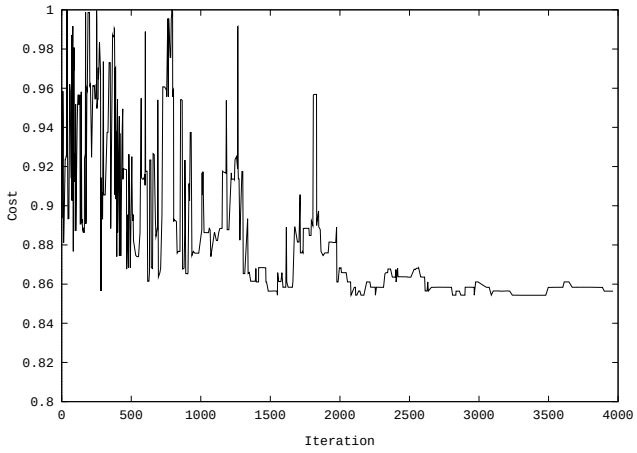

Fig. 3. The cost function for *espresso* (infinite cost excluded from plot).

seen, an immediate attempt is made to reduce the power from that of the initial design, m_0. Although the new power is better than the original, the search continues for a more global minimum. The search is initially liberal in its design selections but eventually settles into a low power region of the design space.

A. Performance of initial designs

Table IV shows the performance of the m_0 designs for the 12 benchmarks. Power consumption has been normalized to the *tomcatv* result. The integer benchmarks achieve lower performance, in general, than the floating-point benchmarks. Execution of integer code also consumes less power by approximately 40% on average. Floating-point units consume higher amounts of power than that of integer units, due to a higher number of transistors per unit. Note also the strong correlation between high IPC/low CTPI and high power usage: more instructions executing in parallel implies more functional units

TABLE IV

PERFORMANCE OF INITIAL DESIGNS.

Class	Benchmark	IPC	CTPI	Power consumption (normalized)
Integer	**compress**	2.18	6.88	0.39
	eqntott	2.23	6.73	0.39
	espresso	2.09	7.17	0.38
	gcc	1.84	8.17	0.36
	li	2.16	6.95	0.37
	sc	2.10	7.15	0.49
	Average:	2.10	7.18	0.40
Floating point	**doduc**	2.65	5.66	0.68
	hydro2d	3.21	4.68	0.63
	mdljdp2	2.36	6.35	0.55
	ora	1.80	8.34	0.51
	tomcatv	3.41	4.40	1.00
	wave5	2.81	5.33	0.61
	Average:	2.70	5.79	0.66

active.

B. Optimized designs

The optimized designs for each benchmark are presented in this section. Four CTPI budgets are considered: 105%, 110%, 120% and 150% of the initial CTPI(m_0). The CTPI, IPC and relative decrease in power consumption values are also presented. The CTPI and power dissipation of the designs are presented graphically in Figures 4 and 5, respectively. Figure 4 shows several interesting trends. The integer benchmark designs do not sacrifice considerable performance except for the 150% budget (recall that lower CTPI is a figure of merit). The 110% designs achieve performance comparable to the initial designs for *espresso, gcc*, and *sc*, while achieving reductions in power. A similar result occurs for *hydro2d, mdljdp2*, and *tomcatv*. A slightly less impressive result can be seen for the remainder of the optimized designs. This demonstrates that the tradeoff technique is successful in finding lower-power yet high-performance designs.

The 150% designs are clearly different from the other designs. These achieve considerable power consumption savings (Figure 5) but at the cost of considerably less performance (Figure 4). The results for *sc* are typical of this phenomenon. The power consumption is reduced by nearly 38%, but for an increase of 66% in CTPI from 7.15 to 10.73.

Tables V and VI present the specific optimized designs for CTPI budgets of 105% (Table V(a)), 110% (Table V(b)), 120% (Table VI(a)), and 150% (Table VI(b)). The tables also present the IPC, CTPI and the percentage reduction in power consumption over m_0 (Table IV) for the optimized designs. The designs are presented in terms of their per-functional unit n and ℓ parameters.

105% and 110% CTPI budget designs

Designs optimized for 105% and 110% budgets represent applications where power must be reduced, but overall superscalar performance is of prime importance. Such applications would include general-purpose computing and mission-critical embedded systems. The reduction in power consumption of the 105% budget is modest for all benchmark-based designs (2.26%–7.64%), with the exception of *ora* (36.4%). The 110% budget presents similar behavior. The most-common unit to duplicate for both budgets is the integer ALU, followed by the *Load* units. Power is reduced primarily through optimized *Load* pipeline depths. Several designs for the floating-point benchmarks choose to include multiple copies of the floating-point units, in spite of their heavy power burden. This is a result of the high-performance goals of this tradeoff analysis.

120% and 150% CTPI budget designs

The 120% and 150% budget designs represent different design goals from the 105%/110% budget designs. Here the goal is to trade superscalar performance for reduced power consumption. An example application would be a low-power embedded system. The 120% budget designs for the integer benchmarks (Table VI(a)) do not differ considerably from the 105%/110% designs. This is not the case for the 150% budget designs (Table VI(b)), where duplicated $IALU$ units have been eliminated and CTPI has increased for four of the six integer benchmarks. The effect of this change on power consumption is dramatic, with power reductions of 32.8%–37.8%. Two exceptions are for *compress* and *eqntott*. The

IALU units are retained and the power reduction is much less impressive. This clearly shows that optimization of the *IALU* unit is critical for low-power embedded systems that execute primarily integer code.

The floating-point benchmarks force several interesting tradeoff decisions for the 120% and 150% budgets. The most-interesting of these is the method chosen for power reduction of the floating-point hardware. The number of floating-point units is reduced over that of the 105% and 110% budget designs, but several benchmarks continue to use duplicated units (e.g., *doduc, hydro2d, tomcatv*, and *wave5*). The power is reduced by decreasing the degree of pipelining from six to four stages for the *FPMul* units and from five to four stages for the *FPAdd* units. Such reductions are reflected in higher CTPI, but the improvements in power consumption are significant. For example, the 4.68% power reduction of the 110% budget *mdljdp2* design (Table V(b)) improves to 26.24% for the 150% design. Other floating-point specific designs achieve lower power by eliminating duplicated *IALU* units.

When combined, these results show that low power designs can be achieved by judiciously adjusting processor organization for power reduction.

IV. Conclusion

This study has presented new techniques for high-level tradeoff analysis and system-level modeling of power consumption before circuit implementation. 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 near-optimal search for organizational tradeoff analysis.

An example cost model was developed to demonstrate the technique and applied to two application areas: high-performance, power optimized designs (105% and 110% CTPI budgets) and embedded, low-power designs (120% and 150% budgets). Several insights can be drawn from the results. Overall power consumption can be reduced via organizational changes alone. For high-performance designs, the techniques in this paper find significant reduction in power for little performance penalty. This result argues for the use of these techniques before the circuit design is commenced. For embedded, low-power designs, two specific trends emerged. For the floating-point applications, the degree of pipelining is a critical parameter. For several integer-intensive applications, the *IALU* unit is the most

critical for power consumption. Although this is an intuitive result, it is not universally true. Two applications (*compress* and *eqntott*) did not eliminate $IALU$ unit duplication, even when performance was allowed to reduce by as much as 50%. This suggests that some applications require higher power designs.

Two extensions to this work are possible. One is the study of additional benchmarks. In particular, power consumption via organizational adjustment is an application-specific task. The methods presented in this paper can be used to study any application. An additional extension is to consider different example technology power consumption models. Both extensions are readily achieved with only minor changes to the overall framework.

Acknowledgments

This research has been supported by AT&T Corporation.

References

[1] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units", *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 34–53, Jan. 1967.

[2] M. Butler, T-Y Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow, "Single instruction stream parallelism is greater than two", In *Proc. 18th Ann. International Symposium Computer Architecture* [23], pp. 276–286.

[3] D. Alpert and D. Avnon, "Architecture of the Pentium microprocessor", *IEEE Micro*, vol. 13, no. 3, pp. 11–21, June 1993.

[4] E. McLellan, "The Alpha AXP architecture and the 21064 processor", *IEEE Micro*, vol. 13, no. 3, pp. 36–47, June 1993.

[5] S. P. Song and M. Denman, "The PowerPC 604 RISC microprocessor", Tech. Rep., Somerset Design Center, Austin, TX, Apr. 1994.

[6] S. Weiss and J. E. Smith, *POWER and PowerPC*, Morgan Kaufmann, San Francisco, CA, 1994.

[7] T. Yeh, *Two-level adaptive branch prediction and instruction fetch mechanisms for high performance superscalar processors*, PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 1993.

[8] S. Weiss and J. E. Smith, "Instruction issue logic for pipelined supercomputers", *IEEE Trans. Comput.*, vol. C-33, no. 11, pp. 1013–1022, Nov. 1984.

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

[10] P. W. Markstein, "Computation of elementary functions on the IBM RISC system/6000 processor", *IBM Journal of Research and Development*, vol. 34, no. 1, pp. 111–119, Jan. 1990.

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

[12] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization", *IEEE Trans. VLSI Systems*, vol. 2, no. 4, pp. 437–445, Dec. 1994.

[13] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design", *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[14] K. M. Dixit, "CINT92 and CFP92 benchmark descriptions", *SPEC Newsletter*, vol. 3, no. 4, 1991, SPEC, Fairfax, VA.

[15] Richard M. Stallman, *Using and porting GNU CC*, Free Software Foundation, Inc., 1989.

[16] 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.

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

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

[19] 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.

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

[21] D. W. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M Powers, "The IBM system/360 model 91: Floating point execution unit", *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25–33, Jan. 1967.

[22] T. Asprey, G. S. Averill, E. DeLano, R. Mason, B. Weiner, and J. Yetter, "Performance features of the PA7100 microprocessor", *IEEE Micro*, vol. 13, no. 3, pp. 22–35, June 1993.

[23] *Proc. 18th Ann. International Symposium Computer Architecture*, Toronto, Canada, May 1991.

The biographies of the authors will be supplied later.

List of Figures

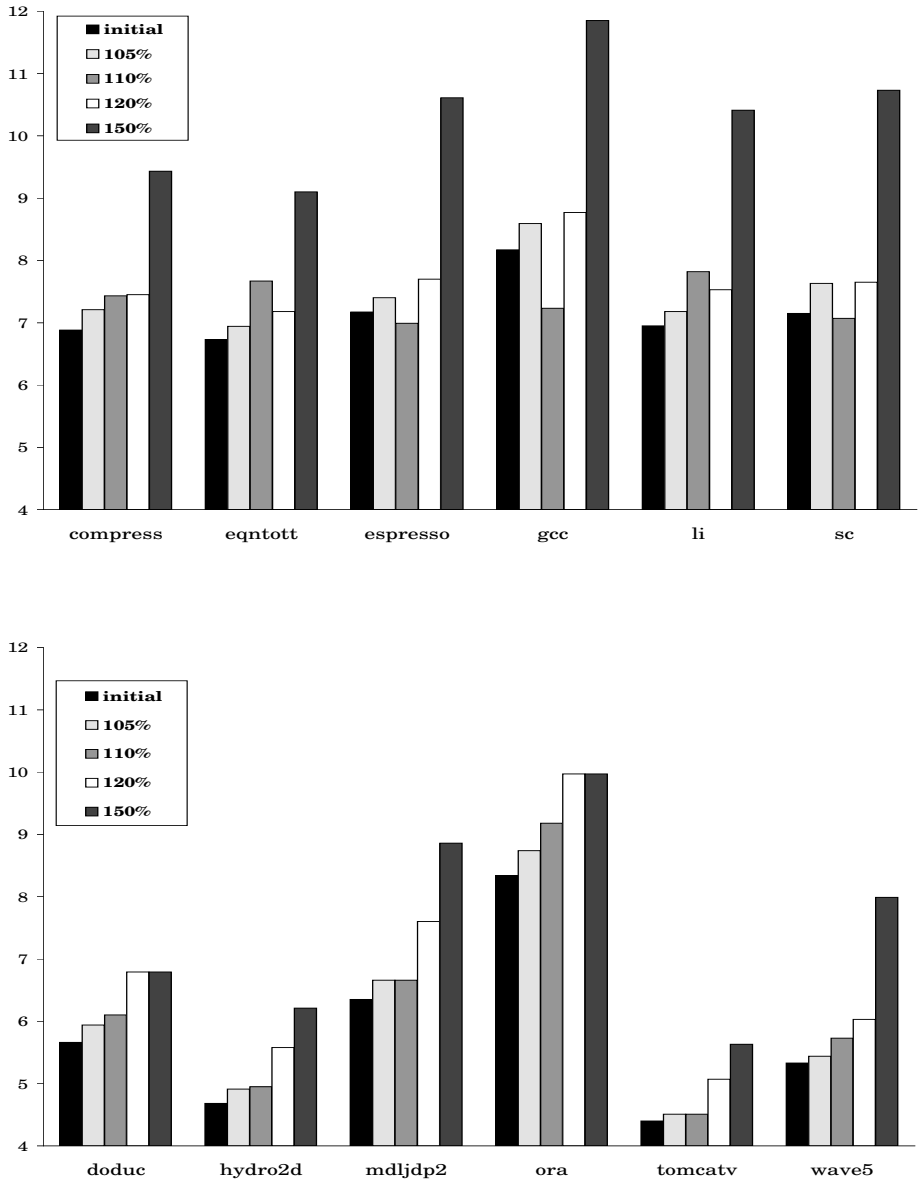

Fig. 4. CTPI for the benchmarks.

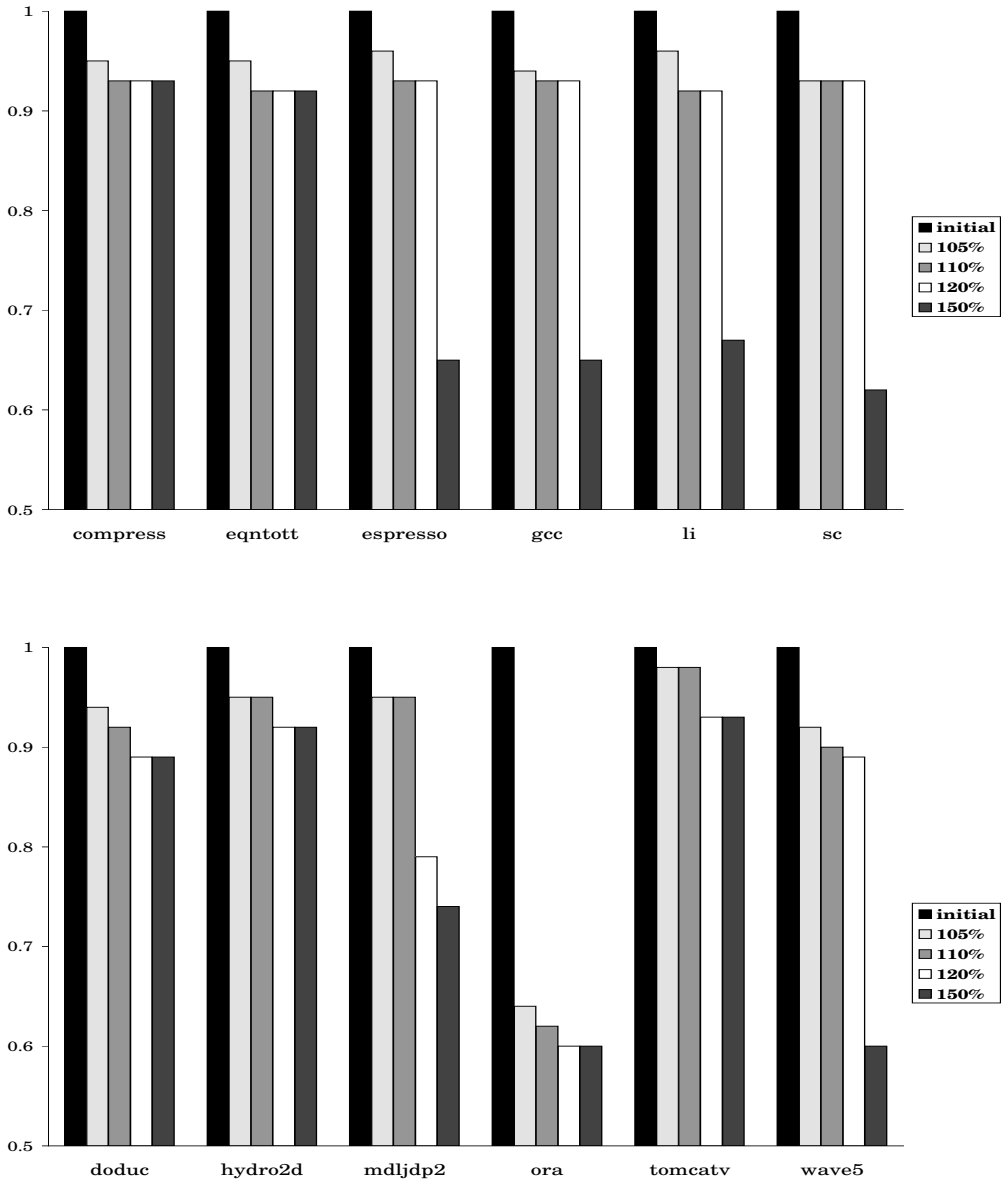

Fig. 5. Power reduction for the benchmarks.

TABLE V

THE LOW-POWER SUPERSCALAR PROCESSOR DESIGNS.

(a) CTPI budget 105%

Benchmark	IPC	CTPI	% Power Reduction	IALU n	IALU ℓ	Shift n	Shift ℓ	Branch n	Branch ℓ	FPAdd n	FPAdd ℓ	FPMul n	FPMul ℓ	Load n	Load ℓ	Store n	Store ℓ	PMiss n	PMiss ℓ
compress	2.08	7.21	5.43	2	1	1	1	1	1	1	5	1	6	2	3	2	1	1	10
eqntott	2.16	6.94	4.67	2	1	1	1	1	1	1	5	1	6	1	3	1	1	1	10
espresso	2.03	7.40	4.43	2	1	1	1	1	1	1	5	1	6	1	3	1	1	1	10
gcc	1.75	8.59	5.67	2	1	1	1	2	1	1	5	1	6	3	4	2	1	2	10
li	2.09	7.18	3.76	2	1	1	1	1	1	1	5	1	6	1	3	1	1	1	10
sc	1.97	7.63	6.80	2	1	2	1	2	1	1	5	1	6	2	4	1	1	1	10
doduc	2.53	5.94	5.79	2	1	1	1	1	1	1	5	1	6	1	2	1	1	1	10
hydro2d	3.06	4.91	4.73	2	1	2	1	1	1	1	5	2	6	4	4	1	1	2	10
mdljdp2	2.25	6.66	4.69	2	1	1	1	1	1	1	5	2	6	1	4	1	1	1	10
ora	1.72	8.74	36.40	1	1	1	1	1	1	3	5	3	6	2	2	3	1	1	10
tomcatv	3.33	4.51	2.26	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
wave5	2.76	5.44	7.64	2	1	1	1	2	1	2	5	1	6	1	2	1	1	1	10

(b) CTPI budget 110%

Benchmark	IPC	CTPI	% Power Reduction	IALU n	IALU ℓ	Shift n	Shift ℓ	Branch n	Branch ℓ	FPAdd n	FPAdd ℓ	FPMul n	FPMul ℓ	Load n	Load ℓ	Store n	Store ℓ	PMiss n	PMiss ℓ
compress	2.01	7.45	7.43	2	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
eqntott	2.09	7.18	7.67	2	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
espresso	1.95	7.70	6.99	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
gcc	1.71	8.77	7.23	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
li	1.99	7.53	7.82	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
sc	1.96	7.65	7.07	2	1	2	1	1	1	1	5	1	6	1	4	1	1	1	10
doduc	2.46	6.10	7.78	2	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
hydro2d	3.03	4.95	5.31	2	1	2	1	1	1	2	5	1	6	1	4	1	1	1	10
mdljdp2	2.25	6.66	4.68	2	1	1	1	1	1	1	5	3	6	1	4	1	1	1	10
ora	1.63	9.18	38.18	1	1	1	1	1	1	2	5	1	6	3	4	2	1	1	10
tomcatv	3.33	4.51	2.26	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
wave5	2.62	5.73	9.63	2	1	1	1	1	1	3	5	1	6	1	3	1	1	1	10

TABLE VI

THE LOW-POWER SUPERSCALAR PROCESSOR DESIGNS (CONT.).

(a) CTPI budget 120%

Benchmark	IPC	CTPI	% Power Reduction	IALU n	IALU ℓ	Shift n	Shift ℓ	Branch n	Branch ℓ	FPAdd n	FPAdd ℓ	FPMul n	FPMul ℓ	Load n	Load ℓ	Store n	Store ℓ	PMiss n	PMiss ℓ
compress	2.01	7.45	7.43	2	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
eqntott	2.09	7.18	7.67	2	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
espresso	1.95	7.70	6.99	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
gcc	1.71	8.77	7.23	2	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
li	1.99	7.53	7.82	2	1	1	1	1	1	1	5	1	6	1	4	2	1	1	10
sc	1.96	7.65	7.07	2	1	2	1	1	1	1	5	1	6	1	4	1	1	1	10
doduc	2.51	6.79	11.26	2	1	2	1	1	1	2	5	1	4	1	4	2	1	1	10
hydro2d	3.05	5.58	7.54	2	1	2	1	1	1	2	5	1	4	1	4	1	1	1	10
mdljdp2	1.97	7.60	21.37	1	1	1	1	1	1	2	5	1	6	3	3	2	1	1	10
ora	1.71	9.97	39.95	1	1	1	1	2	1	1	5	1	4	1	4	1	1	1	10
tomcatv	3.35	5.07	6.69	2	1	2	1	2	1	1	5	1	4	1	4	1	1	1	10
wave5	2.49	6.03	11.45	2	1	2	1	1	1	1	5	2	6	1	4	1	1	1	10

(b) CTPI budget 150%

Benchmark	IPC	CTPI	% Power Reduction	IALU n	IALU ℓ	Shift n	Shift ℓ	Branch n	Branch ℓ	FPAdd n	FPAdd ℓ	FPMul n	FPMul ℓ	Load n	Load ℓ	Store n	Store ℓ	PMiss n	PMiss ℓ
compress	2.01	9.43	7.45	2	1	1	1	1	1	1	4	1	4	1	4	1	1	1	10
eqntott	2.09	9.10	7.70	2	1	1	1	1	1	1	4	1	4	1	4	1	1	1	10
espresso	1.41	10.61	34.95	1	1	1	1	1	1	1	5	1	6	1	4	1	1	1	10
gcc	1.27	11.85	34.62	1	1	2	1	1	1	1	5	1	6	1	4	1	1	1	10
li	1.44	10.41	32.82	1	1	1	1	1	1	1	5	1	6	1	3	1	1	1	10
sc	1.40	10.73	37.80	1	1	1	1	3	1	1	5	1	6	2	3	1	1	1	10
doduc	2.50	6.79	11.31	2	1	2	1	1	1	1	5	1	4	1	4	2	1	1	10
hydro2d	3.06	6.21	7.67	2	1	2	1	2	1	2	4	2	4	1	4	1	1	1	10
mdljdp2	1.92	8.86	26.24	1	1	1	1	1	1	1	5	1	4	1	4	1	1	1	10
ora	1.70	9.97	39.96	1	1	1	1	1	1	1	5	1	4	1	4	1	1	1	10
tomcatv	3.37	5.63	7.10	2	1	1	1	2	1	1	4	1	4	1	4	1	1	1	10
wave5	1.88	7.99	40.22	1	1	1	1	2	1	3	5	1	6	1	2	1	1	1	10