Versatility and VersaBench: A New Metric and a Benchmark Suite for Flexible Architectures

Rodric M. Rabbah, Ian Bratt, Krste Asanovic, and Anant Agarwal
Computer Science and Artifical Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

For the last several decades, computer architecture research has largely benefited from, and continues to be driven by ad-hoc benchmarking. Often the benchmarks are selected to represent workloads that architects believe should run on the computational platforms they design. For example, benchmark suites such as SPEC, Winstone, and MediaBench, which represent workstation, desktop and media workloads respectively, have influenced computer architecture innovation for the last decade. Recently, advances in VLSI technology have created an increasing interest within the computer architecture community to build a new kind of processor that is more flexible than extant general purpose processors. Such new processor architectures must efficiently support a broad class of applications including graphics, networking, and signal processing in addition to the traditional desktop workloads. Thus, given the new focus on flexibility demands, a new benchmark suite and new metrics are necessary to accurately reflect the goals of the architecture community. This paper thus proposes (*i*) VersaBench as a new benchmark suite, and (*ii*) a new Versatility measure to characterize architectural flexibility, or in other words, the ability of the architecture to effectively execute a wide array of workloads. The benchmark suite is composed of applications drawn from several domains including desktop, server, stream, and bit-level processing. The Versatility measure is a single scalar metric inspired by the SPEC paradigm. It normalizes processor performance on each benchmark by that of the highest-performing machine for that application. This paper reports the measured versatility for several existing processors, as well as for some new and emerging research processors. The benchmark suite is freely distributed, and we are actively cataloging and sharing results for various reference processors.

1. INTRODUCTION

Advances in VLSI technology have spurred a shift away from a single-minded focus on performance, and toward new kinds of “all-purpose” architectures. The new architectures are expected to compete with extant microprocessors not in the raw performance they can deliver in specific classes of applications, but rather, in their *versatility*, or the ability to deliver consistently high performance in a very broad set of application domains, from server workloads, to desktop computing and embedded systems. The shift toward versatile architectures is arguably due to the opportunity created by exponentially increasing chip-level resources, combined with the physical limits of power and wire-delay faced by today’s high-performance processors. A recent *New York Times* article [9] reported that Intel for example is revamping its processor road map because chips, such as the widely deployed Pentium 4, have reached their performance limit for the amount of power they consume. Along with other leading companies in the microprocessor industry, Intel is likely to focus on exploiting the characteristics of new and increasingly pervasive application workloads, including graphics, wireless processing, networking, and various forms of signal processing. In these applications, characterized by increasingly prevalent computation paradigms such as *stream processing*, the extraction of parallelism will not require heroic efforts, and as a result, future processors can deliver high performance with significantly lower power costs.

The shift toward versatile architectures can also be observed in the academic arena. Projects such as Smart Memories [20] at Stanford, CDE [1] at UPC, WaveScalar [33] at U. Washington, TRIPS [27] at UT Austin, Raw [39] and SCALE [15] at MIT, have all stated as a goal the ability to efficiently run a broad class of applications, including those from traditional desktop and embedded system domains. The recent DARPA program in Polymorphic Computing Architectures [25] is also a research thrust in this new area marked by architectural versatility.

Motivated by the focus on versatility, this paper proposes a new benchmark suite called *VersaBench*, and a new metric called *Versatility* to accurately reflect the new goals of a significant portion of the architecture community. Much as the Standard Performance Evaluation Corporation (SPEC) [8] has influenced the design of general purpose CPUs in the past, we will demonstrate how the Versatility metric along with the VersaBench suite may guide the design and optimization of future all-purpose computers.

MIT-LCS-TM-646, June 2004, Pages 1–17.

The VersaBench suite is a collection of fifteen benchmarks that are grouped into five application categories encompassing desktop integer workloads, floating-point and scientific applications, server computing, stream processing, and embedded bit-level computation. The VersaBench applications are drawn from various suites, and were selected because of the salient behavior and the properties they exhibit along various dimensions. In all, we consider five *basis* property-dimensions when characterizing a benchmark, including *parallelism*, *instruction temporal locality* (which is the inverse of control complexity), *data temporal locality*, and *data spatial locality*. The VersaBench suite was created systematically by measuring the properties of numerous applications, and selecting those that best matched the properties that intuitively describe applications from the five central computing tiers—desktop integer, desktop floating-point and scientific, servers, embedded streaming, and embedded bit-level systems. The constituents of the VersaBench suite are listed in Table 2 and described in Section 3. It differs from previous benchmark suites such as MediaBench [16], Winstone [42], and SPEC in two ways. First, it uses a new Versatility metric, and second, it consists of applications drawn from five computing classes; whereas Winstone, SPEC and MediaBench are heavily weighted toward desktop computing, workstation workloads and streaming media respectively.

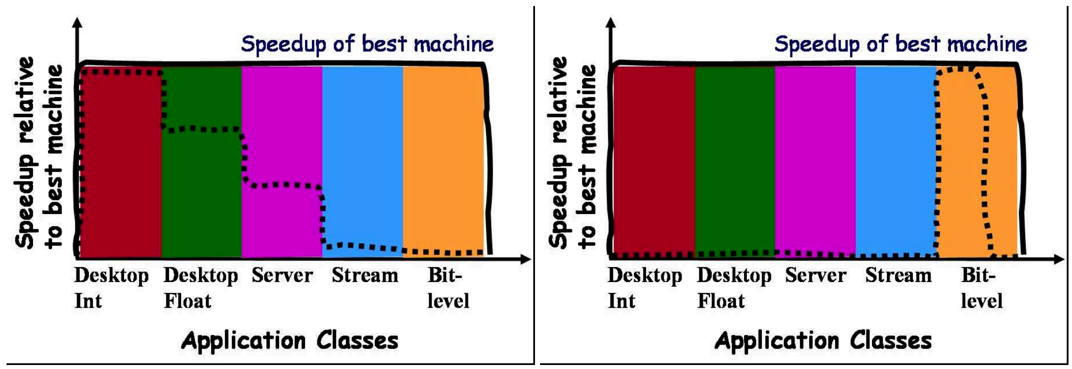
The Versatility metric is inspired by the SPECmark metric which is the de facto standard for reporting the throughput of an architecture for a given SPEC benchmark suite. A SPECmark provides a scalar measure to compare architectures, and it may be normalized by power, area, or cost to reflect an efficiency along dimensions other than performance. Specifically, the SPEC CINT89 SPECmark for an architecture is the geometric mean of the speedups of that architecture relative to a reference machine (specifically, the VAX 11/780)[1] for each of the applications in the SPEC CINT89 suite.

Computing the Versatility of an architecture purposefully mirrors that of a SPECmark. Accordingly, Versatility is a scalar quantity defined as the geometric mean of the speedups of an architecture for each of the applications in the VersaBench suite. Unlike SPECmarks however, the speedup of each application is not computed relative to a single reference machine, but rather relative to the architecture which provides the *best* performance for that application (in the 2004 time frame from known results at the time of this writing). A formal definition of Versatility can be found in Section 2. Like SPEC, the Versatility may be separately normalized by chip area, power or machine cost.

Intuitively, Versatility measures how close the performance of an architecture is to that of a hypothetical, idealized versatile machine (IVM) whose performance on each application equals that of the highest-performing machine (HPM) for that application. Since a single scalar metric can hide a lot of information, additional insight into the strengths and weaknesses of an architecture can be obtained by going beyond the scalar metric and graphing the individual speedups for each application relative to the HPM for that application, as shown in the abstract graphs in Figure 1. Points on the X-axis represent various benchmark applications, and points on the Y-axis represent speedups compared to the HPM for each application. The solid line *envelope* in Figure 1(a) represents a hypothetical ideal versatile machine, or IVM. The dotted line in (a) might represent a conventional general purpose processor (e.g., a Pentium 4) whose performance for desktop integer applications equals that of the HPM for those applications, while its performance for streaming computations falls short of the best achievable by a specialized vector or stream engine such as VIRAM [14] or Imagine [12]. The dotted line in Figure 1(b) depicts the characteristics of an ASIC (application-specific integrated circuit) which performs exceptionally well for a few bit-level applications, but is unable to run other types of programs. Clearly, a highly versatile architecture that does well across the board will closely track the IVM envelope. Because we will reference such graphs frequently, we name them *VersaGraphs.*

VersaGraphs help identify the application areas where opportunities for architectural improvements are greatest. For example, a new architecture that performs as well as a Pentium 4 for desktop integer workloads has little to gain from further improvements targeted toward that application domain. In contrast, an architecture that fares poorly compared to the best stream processor warrants attention that is focused on improving the performance of that architecture within the streaming context.

[1] The reference machines have changed over time. While the VAX 11/780 was the reference machine for SPEC CINT89 and SPEC CINT92, the SPARCstation 10/40 was the reference machine for SPEC CINT95, and the Sun Ultra5-10 workstation with a 300MHz SPARC processor is the reference machine for SPEC CINT2000.



(a) (b)

Figure 1. Example VersaGraphs. The solid-line outer envelope in (a) and (b) represents a hypothetical ideal versatile machine, or IVM. The dotted line in (a) might represent one of today's general purpose processors. The dotted line in (b) represents a typical ASIC.

1.1 Summary of Contributions

In summary, this paper makes the following contributions:

— A new benchmark suite picked from five applications classes. The suite is geared toward the characterization of architectural flexibility. We are maintaining the suite on the web, and freely distributing it, including benchmark implementations in C, and where relevant, in Verilog, a stream language, and assembly code.

— A new metric called Versatility. This metric provides intuitive, normalized results that quantify the flexibility of ASICs, general purpose processors, and tiled processor architectures, even as their performance improves over time.

— The paper presents results to demonstrate that the characteristics of the five application classes are distinct and measurable.

— The paper characterizes the Versatility of several extant commercial and research processors. We are also actively cataloging and sharing results for our reference processors and machines.

1.2 Roadmap

The remainder of this paper is structured as follows. Section 2 introduces the Versatility metric, Section 3 discusses the VersaBench suite and the properties of the constituent applications. Section 4 describes the benchmarking methodology, and Section 5 reports our results for several architectures using VersaBench and the Versatility metric. Section 6 discusses related work, and Section 7 summarizes and concludes the paper.

2. THE VERSATILITY METRIC

We define the Versatility of an architecture as the geometric mean of the speedup of each of the applications in the VersaBench suite relative to the architecture which provides the best execution time for that application.

In comparison, the SPECmark of an architecture is the geometric mean of the speedup of each of the applications in a specific SPEC suite (e.g., SPEC INT 89) relative to the execution time of the application on a *single* reference machine (the VAX11/780 for SPEC INT 89). Like SPEC, Versatility provides several other results that are interesting in their own right as summarized in Table 1. Table 1 further highlights the similarities between the two metrics and shows that they differ only in how the reference runtime is defined.

Table 1. VersaBench versus the SPEC metric. They differ only in how they define the reference runtime.

VersaBench	SPEC INT 89
Application runtime on processor	Application runtime on processor
Application reference runtime chosen as best known runtime in 2004	Application reference runtime chosen as runtime of a VAX 11/780
Speedup for application over reference runtime	Speedup for application over reference runtime
Geometric mean of speedups over all applications	Geometric mean of speedups over all applications

The SPEC metric is useful when comparing the performance of two architectures for workstation workloads. The Versatility metric can also be used to compare the performance of two architectures across the diverse VersaBench suite. In addition, the Versatility metric of an architecture is indicative of how close its performance is to that of an IVM, in other words, a hypothetical processor whose performance on each application equals that of the HPM for that application.

More formally, suppose VersaBench has N benchmarks numbered $1,2,3,\cdots i,\cdots N-1,N$. In our suite, $1-3$ are desktop integer benchmarks, $4-6$ are desktop floating-point applications, $7-9$ are server benchmarks, $10-12$ are embedded streaming applications, and $13-15$ are embedded bit-level programs.

Next, suppose that the running time for application i on architecture A is denoted as A_i, and further, the running time achieved by the HPM for i is denoted as R_i (i.e., this is the reference time for application i on the highest-performing machine for that application). Mathematically,

$$R_i = \min_{\forall A}(A_i)$$

Note that R_i may be the same as the corresponding A_i if A is the HPM for application i. The speedup of architecture A relative to the reference time (i.e., the best execution time) for application i is given by the ratio R_i/A_i. Finally, the Versatility of architecture A is

$$V_A = \left[\Pi_{i=1}^{N} \frac{R_i}{A_i}\right]^{\frac{1}{N}} \tag{1}$$

or simply the geometric mean of the speedups. Interestingly, Equation 1 also defines the calculation of the SPEC metric, with the distinction that R_i for SPEC is the running time of the VAX 11/780 for application i.

2.1 Rationale and Remarks

We chose to use the geometric mean to be consistent with existing benchmarking practices, although it is well known that the harmonic mean of speedups is a better metric when the total execution time is the primary concern [31]. Unlike the arithmetic mean, both the geometric and harmonic means for speedups have the intuitively appealing property that an architecture's Versatility is zero if it is unable to run even a single application; thus the Versatility of an ASIC is zero.

The Versatility metric can be used in one of two ways when considering the evolution of machines and their performance. In one way, we select the best machines for each application at the time of this writing (in 2004), and we always use their respective running times to normalize speedups. For example, if at the time of this writing, a 3 GHz Pentium 4 is the HPM for application i, then we will always normalize A_i relative to the Pentium 4's runtime for i. This approach provides a common standard to measure flexibility for all time. However, the drawback is that as machines get faster over time, their Versatility will eventually surpass one. This is not counter intuitive however, since faster processors can run more applications effectively. The SPEC analogy is to normalize to the performance of a VAX 11/780 for all time.

In an alternate approach, we can renormalize to a new set of HPMs every year or every few years, so that the Versatility of machines is always below one. The process of renormalizing numbers for older machines is easy with geometric means because it does not require a knowledge of the individual application speedups. Specifically, if there are N applications in the VersaBench suite, and the best execution time for one application decreases by a factor β, then the Versatility of any architecture will decrease by a factor of $\beta^{(1/N)}$. This result follows directly from Equation 1.

We do not take a position on the proper method for the evolution of the Versatility metric over time. Instead, we propose to maintain two sets of Versatilities for each machine: one that keeps the normalizing running-times fixed for all time—this is V_A—and another that renormalizes every year—this is Y–V_A, where Y is the year in which the best machines are chosen. Thus, for example, if the Versatility of an architecture A is 0.5 in 2004, and is 0.2 when normalized to HPMs in 2006, then the Versatility of A is specified either as 0.5 or as 2006–0.2. Note that V_A and 2004–V_A are one and the same because we are starting in the year 2004.

Our metric weights the application classes equally. However, the metric can easily be customized to use relative weights. As an example, if a company wishes to produce a chip that is highly optimized for streaming workloads—while providing barely adequate performance for desktop applications—the following variant of Versatility can be used:

$$V_A = \left[\Pi_{i=1}^{N}\left(\frac{R_i}{A_i}\right)^{w_i}\right]^{\frac{1}{\Sigma_{i=1}^{N} w_i}}$$

Also note that the *Versatility metric, and its variants, can be used with other benchmark suits that may reflect applications requirements other than the ones considered in this paper*.

2.2 Improving Versatility

Intuitively, improvements in Versatility can come from two sources: (i) absolute increase in performance, and (ii) better performance robustness across all applications. In terms of the former, performance boosts can come via advances in technology that allow the same architecture to run with a faster clock. Faster processors can run more applications effectively and are therefore more versatile. Notice that as general purpose processors became faster over the years, they were able to process audio and many forms of video in real time, thereby reducing their need for special purpose hardware.

In terms of performance robustness, a processor is highly versatile if it can run an entire range of applications effectively. The Versatility metric conveys quantitatively how close the performance of an architecture is to that of an ideal versatile machine[2], and identifies potential opportunities for architectural improvements. Namely, the normalized speedups of the individual benchmarks can be plotted in the form of a VersaGraph to identify the application classes for which a machine has room for improvement, and classes for which the architecture is already doing as well as can be expected in the given timeframe. Thus while the Versatility equation can be factored as

$$V_A = \left[\Pi_{i=1}^{N} R_i\right]^{\frac{1}{N}} \left[\Pi_{i=1}^{N} \frac{1}{A_i}\right]^{\frac{1}{N}}$$

where the first term is independent of the architecture A (i.e., it is a common factor in the versatility computations of all architectures), the normalization serves dual purposes as just described, and is therefore not gratuitous.

In the following Section, we describe the Versatility benchmark suite, and in Section 5, we present VersaGraphs for several architectures, along with their measured Versatilities and other related analysis.

3. VERSABENCH

Benchmark suites are assembled to represent the workloads that architects believe should run on the computational platforms they design [18]. For example, benchmark suites such as SPEC, Winstone [42], and MediaBench [16]—which

[2]The Versatility (in 2004) of an IVM is unity, as its performance on each application equals that of the HPM for each application.

Table 2. The VersaBench suite.

Desktop Integer	`mcf`	Combinatorial optimizer
	`parser`	Link grammar parser
	`twolf`	Place and route simulator
Desktop Floating Point	`bmm`	Blocked matrix multiplication
	`vpenta`	Pentadiagonal matrix inverter
	`tomcatv`	Vectorized mesh generator
Server	`mesa`	3D-Graphics library (24 copies)
	`mgrid`	Multi-grid solver (24 copies)
	`dbms`	DataBase Management System (24 copies)
Embedded Streaming	`corner`	Large matrix transposer
	`fm`	Software FM radio
	`beam`	Multi-channel beam-former
Embedded Bit-Level	`80211a`	Convolutional encoder from IEEE 802.11a Standard
	`8b10b`	IBM 8bit/10bit block encoder
	`des`	ECB encoder of the Data Encryption Standard

represent workstation, desktop, and embedded workloads respectively—have driven computer architecture innovation for several years. Recently, several projects have begun to investigate more flexible architectures that can efficiently support a broader class of applications including graphics, networking, and signal processing in additional to the traditional workstation and desktop workloads. Thus, given the new focus on versatility, new benchmark suites must also evolve to reflect the broader application requirements.

This section describes VersaBench, our proposed benchmark suite to facilitate the versatility measurements of next-generation architectures. The benchmark suite consists of fifteen benchmarks that are grouped into five categories: *desktop integer* (INT), *desktop floating-point* (FLT), *server* (SVR), *embedded streaming* (STR), and *embedded bit-level* (BIT), with the first two categories reflecting traditional ILP-centric sequential workloads. The benchmark suite is freely distributed, and we believe the community will help evolve the collection to continuously reflect the range of workloads architects wish to run on the processors they design. The VersaBench suite is easy enough to run and we encourage researchers to evaluate their architectures and report their results to the community via our cataloging website.

The VersaBench constituents are briefly described in Table 2. They are drawn from various suites, and were selected because of the salient properties they exhibit along various dimensions. For example, desktop applications are expected to have complex control structures, and hence poorer instruction-temporal locality, whereas streaming applications consist of relatively small computational kernels with simple control mechanisms, and hence better instruction temporal locality. In what follows, we present simple measures to quantify *control complexity*, as well as several other fundamental dimensions of interest. In all, there are five property-dimensions, and they are

— *predominant data type*: summarizes the predominant type-domain over which computation is performed,

— *parallelism*: quantifies maximum IPC (instructions per cycle) in a benchmark,

— *instruction temporal locality* (ITL): measures the degree to which instructions are repeated in a given time window, and is related to the inverse of the control complexity,

— *data temporal locality* (DTL): captures the degree to which data addresses are repeated in a given time window, and

— *data spatial locality* (DSL): captures the degree of address adjacency in data traces.

Intuitively, we believe the properties of each of the five benchmark-categories are as shown in Table 3. Accordingly, the VersaBench suite was created systematically by measuring the properties of numerous applications and selecting those that matched expectations. Namely, we favored desktop integer benchmarks with low parallelism, ITL, and DSL, high DTL, and of course those which predominately consisted of computation using integer data types. Similarly, for the

Table 3. Characteristics of the VersaBench workloads.

	Type	**IPC**	**ITL**	**DTL**	**DSL**
Desktop INT	integer	low	low	high	low
Desktop FLT	float	medium	medium	medium	medium
Server	integer/float	high	low to medium	medium to high	low to medium
Streaming	integer/float	very high	high	low to high	very high
Bit-Level	bit	medium-high	very high	low	very high

desktop floating-point benchmarks, we chose benchmarks that are floating-point heavy, with relatively higher parallelism, ITL and DSL, and relatively lower DTL. It is expected that scientific applications in the desktop floating-point category are more amenable to compiler optimizations (e.g., loop unrolling) that expose parallelism. Such applications usually process large quantities of data and hence have lower data temporal locality compared to the integer benchmarks. The server-oriented benchmarks often exhibit properties that mimic integer and floating-point workloads, although they have substantially more parallelism by way of concurrent processes.

In the embedded domain, the streaming benchmarks largely represent signal processing computation. They are characterized by kernels that process potentially infinite sequences of short-lived data. Often, stream applications also *peek* at future data items before they are processed, as in the computation of moving averages. They may also perform substantial coefficient lookups, and as a result, stream benchmarks exhibit varying degrees of temporal locality. When stored in temporary memory buffers, stream data addresses exhibit high data spatial locality. In addition, the kernels in streaming systems are often self-contained and have well defined static control structures. Hence, they are massively parallel and have high instruction temporal locality. The bit-level benchmarks are similar in terms of their characteristics, although they may carry out complex bit-twiddling and state updates that limit parallelism.

The following sections describe our methodology for quantifying the property-dimensions of the VersaBench suite. We initially focus on each application-class as a whole and illustrate that the basis properties are distinguishable. The goal was to demonstrate that each of the domains can be characterized by one or more basis properties. For example, the desktop integer applications have high control complexity (and hence low ITL) and are limited in terms of IPC (Table 3). By contrast the streaming benchmarks have significantly more parallelism and different locality behaviors. Here, we report the properties for the individual benchmarks.

3.1 Data Types

The predominant data type in an application dictates the nature of the computation, with distinctions drawn between integer arithmetic, floating-point arithmetic, and bit-level processing. Applications from each of the five VersaBench categories intrinsically consist of different predominant data types. We identify the predominant data type and computation by manually inspecting the source code of the application. In addition, we measure the dynamic instruction mix of each benchmark using a cycle accurate simulator, and summarize the results in Figure 2. There are five main types of instructions: branch, compare, memory, integer, and floating point. The control intensive applications will have a much large percentage of compare and branch instructions. Computation intensive benchmarks will have a larger percentage of integer or floating point operations: nearly 40% of the operations in the desktop integer applications are compare and branch operations; compare this to the 70-80% of operations spent in actual computation in the bit-level benchmarks. The desktop floating-point programs are the most memory intensive and stress the floating point unit with roughly 20% of their instructions. The instruction mix for the server benchmarks shows they are slightly more control-complex compared to the floating-point applications, and do not stress the floating-point ALU as heavily. The streaming benchmarks have a range of instruction mixtures with respect to the integer versus floating-point operations. They also vary with respect to the number of memory operations; two of three stream benchmarks have significantly more loads than stores because they perform significant peeking.

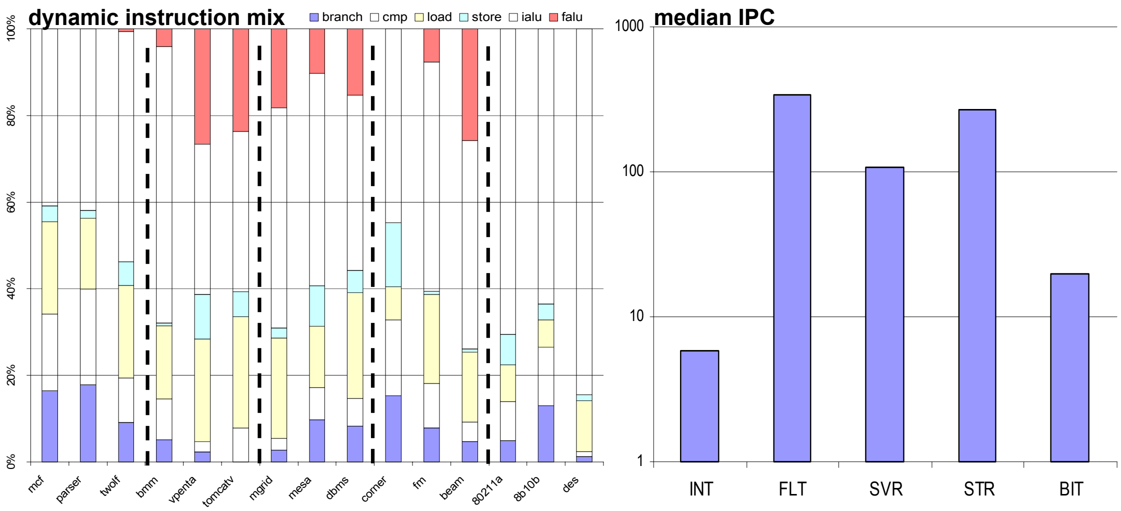


Figure 2. **Left.** Dynamic instruction mix for the five application-classes in VersaBench. The classes are listed in following order: INT, FLT, SVR, STR, and BIT. The classes are separated by the dashed vertical lines. **Right.** The median IPC for the VersaBench suite; note that a log-scale is used.

3.2 Parallelism

The intrinsic parallelism of an application is an important property that varies across workloads. It is expected that embedded applications will exhibit massive levels of parallelism, whereas desktop integer application are quite limited in this regard.

To measure parallelism, we focus on instructions per cycle (IPC), and perform a "limit study" to discover how much parallelism exists in the application. Specifically, given a sequential instruction trace, we dynamically schedule the operations assuming infinite resources (i.e., unlimited number of registers and functional units) and oracular knowledge (i.e., perfect branch prediction and memory disambiguation). Hence all artificial dependences are removed, much like the ILP limit studies performed by Wall [40]. We also heavily optimize each benchmark and perform aggressive inlining and loop unrolling to break legacy performance barriers. Our compilation and simulation environment is based on the Trimaran research infrastructure [37]. Trimaran is a publicly available compilation and simulation framework that includes a number of high level optimizations geared toward increasing ILP (e.g., superblock [11] and hyperblock [19] optimizations). In our "perfect" processor, we execute instructions as early as possible, limited only by dataflow dependences. Also, while we assume perfect branch prediction, branch instructions are not free and they contribute to the critical path length of the execution.

In Figure 2, we report the median IPC for each application-class from the VersaBench suite (detailed results for each benchmark can be found in Table 4). The IPC is the average instruction issue rate, assuming all instructions have a one-cycle latency. The results show that the desktop integer benchmarks ranked lowest in terms of IPC, largely due to their control complexity. Streaming benchmarks occupy the other extreme, partly because of their lower control complexity, and partly because there is significant task level parallelism that is exploitable. The bit-level applications have lower parallelism by comparison, despite their very low control complexity. This is because of the significant number of state updates that are necessary for "bit-twiddling". The desktop floating point and server workloads both have substantial parallelism. In the case of the server workloads, the amount of IPC can potentially increase further as the number of concurrent threads is increased (we assume twenty four concurrent threads in VersaBench). Thus, the server applications in VersaBench can benefit from support for multithreading in processors. On a related note, we do not distinguish between instruction level parallelism (ILP), data level parallelism (DLP), or thread level parallelism (TLP). We believe

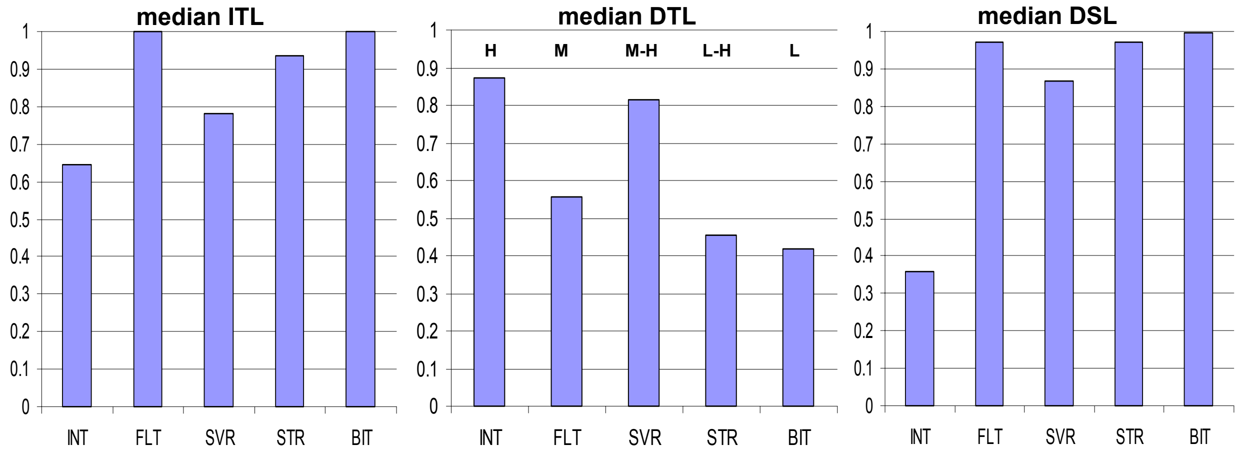


Figure 3. The median IPC (left), DTL (middle) and DSL (right) for the VersaBench application-classes. In the DTL graph, we indicate the expected locality for each of the classes (H: high, M:medium, L:low).

that to a large extent, IPC is a reasonable measure for parallelism of any sort. It is also easiest and most straightforward to measure.

3.3 Instruction Temporal Locality

Instruction temporal locality characterizes the reuse of instruction addresses as a function of time. To measure instruction temporal locality, we extract an address trace $\mathcal{T}$ from the application as it executes, and partition the trace into contiguous and non-overlapping trace segments s_1, s_2, $\ldots$, s_n, such that $\mathcal{T} = s_1|s_2|\ldots|s_n$. When measuring instruction temporal locality, $\mathcal{T}^I$ consists of the sequence of addresses corresponding to the instructions that are (serially) executed by the processor. By construction, each segment s_i contains exactly $\mathcal{W}$ unique addresses, but may vary in length ($l_i = |s_i|$) since an address may occur several times within a segment.[3] The instruction temporal locality of a segment is

$$ITL(s_i) = 1 - \frac{\mathcal{W}}{l_i}$$

or simply one minus the number of unique references in a segment divided by the total number of addresses in the segment. A segment that lacks address reuse therefore does not have any temporal locality (i.e., $\mathcal{W} = l_i$ and hence $ITL = 0$); temporal locality increases proportional to l_i thereafter. The temporal locality for the entire sequence $\mathcal{T}$ is the arithmetic mean temporal locality of all of its segments.

We measured the temporal locality of instruction traces for several values of $\mathcal{W}$ in the range $[2^5, 2^{16}]$. In Figure 3, we plot the median ITL for each of the application-classes using $\mathcal{W} = 2^{10}$. For the most part, the ITL varies little from one class to the next, with the biggest distinction made for the desktop integer suite, as expected. The highest ITL is measured for the bit-level applications, and indeed these benchmarks have the smallest instruction memory footprints and only show variations for very small values of $\mathcal{W}$. The stream benchmarks have comparable ITL, although they have large instruction memory footprints; the streaming kernels can be fully unrolled and can impact the measured ITL up to $\mathcal{W} = 2^{11}$.

[3] $\mathcal{W}$ is similar to the popular notion of working set size.

3.4 Data Temporal Locality

Data temporal locality characterizes the reuse of data addresses as a function of time. It is measured in a manner similar to the instruction temporal locality, using $\mathcal{T}^D$, the trace of data addresses that are (serially) fetched by the processor. To measure data temporal locality, we varied $\mathcal{W}$ in the range $[2^9, 2^{20}]$, and in Figure 3, we plot the median DTL for the five application-classes using $\mathcal{W} = 2^{13}$. The measured DTL values closely follow intuition (Table 3), with integer benchmarks at the high end of the spectrum, and bit-level applications at the lowest. Note that as noted earlier, the streaming benchmarks can widely range in terms of their DTL, proportional to the amount of data-peeking that they may perform. In general, the DTL of a streaming kernel is roughly $1 - \frac{1}{p}$, where p is the kernel peek-rate. The server and floating-point workloads have relatively large data memory footprints. They vary in terms of their processing patterns, from dense matrices, to sparse mesh computations. Their DTL are in the medium range for the most part.

3.5 Data Spatial Locality

Temporal locality represents the reuse of addresses as a function of time, whereas spatial locality describes the clustering of data in space. Specifically, data spatial locality refers to the extent of data address adjacency in an address stream. That is, given two addresses α_1 and α_2, if $|\alpha_1 - \alpha_2| \leq \mathcal{K}$ then the addresses are said to share the same locale. Memory architectures widely exploit spatial locality by fetching a *block* of data at a time from memory to the caches; this amortizes and ameliorates the cost and latency of a memory access, as long as the data within the block is utilized in time (i.e., data within the same spatial locale also share temporal locales).

To measure data spatial locality, we divide the memory space into equal-sized blocks such that every addressable byte is a member of a unique block. Given a trace segment s_i partitioned as described earlier, we map each address in the segment to its parent block, and count the number of unique blocks that are referenced. If $\mathcal{B}$ equals the number of addressable bytes in a block (i.e., $\mathcal{B}$ is the block size), then an address α belongs to block $\alpha \mod \mathcal{B}$. Hence, using $\mathcal{W}_i'(\mathcal{B})$ to represent the number of unique blocks referenced in a segment s_i, data spatial locality is defined as

$$DSL(s_i) = \frac{\mathcal{W}}{\mathcal{W}_i'(\mathcal{B}) \times \mathcal{B}}$$

or simply the ratio of the unique references in s_i to the total number of data in the accessed blocks ($\mathcal{W}_i'(\mathrm{B}) \times \mathcal{B}$). Intuitively, the spatial locality measure quantifies the "pollution" in a block. A segment with "perfect" spatial locality (i.e., $DSL(s_i) = 1$) does not have any pollution because every addressable byte in every accessed block is referenced. In contrast, if every (byte-sized) reference in s_i maps to a different block, then $\mathcal{B} - 1$ bytes per block are not used and considered pollution; note that the minimum value for spatial locality is $DSL(s_i) = \frac{1}{\mathcal{B}}$. The spatial locality for the entire sequence $\mathcal{T}^D$ is the arithmetic mean spatial locality of all segments in the trace.

Data spatial locality varies significantly between workloads, and the median DSL for the VersaBench application-classes is shown in Figure 3. In the plot, we assume $\mathcal{W} = 2^{13}$ and $\mathcal{B} = 32$ bytes. As expected, the embedded benchmarks have the highest levels of address adjacency, as the data they processed is tightly packed in contiguous memory locations. The integer workloads, which are often pointer-heavy, have the lowest levels of DSL.

4. BENCHMARKING METHODOLOGY

Our philosophy is to have simple benchmarks that are easy to run. Some of the VersaBench applications are part of the SPEC suites, and when available, we used the MinneSPEC reduced reference workloads [13]. In terms of Versatility measurements, we propose the following guidelines for future reporting—these are the same rules that were followed for the results reported in this paper:

— Benchmarks may be compiled with the best available compiler.

— Benchmarks may be rewritten in any language (e.g., C, Java, StreamIt [36], Brook [7], Verilog) provided they adhere to the original algorithms. For example, a recoding of `bmm` must use the blocked matrix multiply algorithm. The

Table 4. Measured properties of VersaBench.

Benchmark	IPC	ITL	DTL	DSL
mcf	5.8	0.948	0.483	0.360
parser	4.8	0.645	0.894	0.335
twolf	10.8	0.255	0.871	0.637
bmm	10560.5	1.000	0.817	0.9370
vpenta	339.6	0.415	0.450	0.997
tomcatv	119.5	1.000	0.555	0.972
mgrid	576.48	0.965	0.815	0.949
mesa	108.8	0.781	0.457	0.866
dbms	19.2	0.440	0.854	0.592
corner	43995.1	1.000	0.011	0.944
fm	266.6	0.923	0.988	0.988
beam	76.5	0.934	0.453	0.972
80211a	12.3	1.000	0.418	0.995
8b10b	19.9	1.000	0.322	0.998
des	77.5	0.992	0.803	0.986

benchmarks may even be hand-coded in assembly to suit a particular architecture. The VersaBench website provides several versions for many of the benchmarks.

— The Versatility may be computed using wall clock times (preferred) or number of cycles, as long as the method used is clearly specified.

— Real architectures are preferred, but simulators may be used.

— Although the modeling of real I/O is encouraged, we recognize the difficulty in doing so in a prototype environment. We suggest initializing a region of external DRAM with I/O data, and flushing caches so they are not primed prior to the measurement process. Simulation environments often ignore system calls, in that they are treated as magical instructions that can atomically update memory, without polluting the caches. Alternatively, a *deionizer* [35] may be used to idealize I/O. We went to great lengths to minimize the effects of I/O in the VersaBench suite.

The evaluation process thus affords a lot of flexibility in how the benchmarks may be coded and executed. However, when reporting results, the details of the methodology that is adopted must be clearly described. Some common parameters include (*i*) whether a simulator is used, (*ii*) the language and compiler used in the implementation (and if any hand-coding is done), (*iii*) whether wall clock times are used, or whether cycles are being measured, (*iv*) the clock speeds that are assumed for the architecture, (*v*) whether I/O is accurately simulated, or if the I/O costs are ignored, and (*vi*) the speeds assumed for caches and external memory, and whether the external memory is faithfully modeled.

The next section reports our Versatility measurements for five different reference machines (see Table 5). These machines were chosen primarily to assess the versatility of existing general purpose processors as well as the versatility of an existing tiled-processor architecture (TPA). Physical accessibility was a major factor in choosing machines for this evaluation. In the case of the Pentium III (P3), Pentium 4 (P4), Itanium 2, and Athlon 64, the results were gathered from the real hardware. We measured the timing non-invasively and consistently across all four commercial platforms. In the case of Raw, cycle counts were obtained from a cycle accurate simulator that has been verified against an existing Raw chip. When compiling for the P3, P4, Itanium 2, and Athlon 64 the same optimization flags were used for all benchmarks (Table 5). In keeping with the ideas behind versatility, future work will determine the best optimization flags for each benchmark. For Raw, `rgcc`—a port of `gcc`—was used to compile the integer and server benchmarks. The floating point benchmarks were compiled with `rawcc` [17], the streaming benchmarks were compiled with `strc`—the StreamIt compiler—and the bit-level applications were hand-coded in assembly.

Table 5. Experimental Setup.

Machine	Clock	Process	Cache	Compiler	Compiler Flags
P3	600 MHz	180-nm	L1-D 16K, L1-I 16K, L2-U 256K	`gcc` 3.3	-O3 -march=pentium3 -mfpmath=sse
P4	2.8 GHz	130-nm	L1-D 8K, TC 12k uops, L2-U 512K	`gcc` 3.3	-O3 -funroll-all-loops -march=pentium4 -ffast-math
Itanium2	1.3 GHz	130-nm	L1-D 16K, L1-I 16K, L2-U 512K, L3-U 3M	ecc 7.1	-O3 -ipo
Athlon64	2.2 GHz	130-nm	L1-D 64K, L1-I 64K, L2-U 1M	`gcc` 3.3	-O3 -funroll-all-loops -fmarch=athlon -ffast-math
Raw	425 MHz	180-nm	16*L1-D 32K 16*L1-I 32K	`rawcc/rgcc` and `strc`	-O3

5. RESULTS

We summarize the Versatilities of the five processors in Table 6. We find that the Athlon 64 is the most versatile of the five processors, with a Versatility of 0.458. It is interesting to note that the P3 and Raw are implemented in the same technology, yet their Versatilities are disparate: $V_{P3} = 0.1$, and $V_{Raw} = 0.406$. The TPA outperforms the Pentium 3 on the embedded applications. The TPA approaches the Versatility of the Athlon 64 even though it runs at a much slower clock frequency and is implemented in an older technology.

Table 6. Versatility numbers

	P3	P4	Itanium 2	Athlon 64	Raw
Versatility	.104	.375	.316	.456	.406

The speedups of each architecture relative to the P3 can be seen in Figure 4. The Athlon 64 performs fairly consistently across the board, often outperforming the P4 and the Itanium 2. This is in part due to the fact that the Athlon 64 is the newest machine in the group; we suspect that the latest generation P4 and Itanium processors can rival the Athlon in performance. The Raw processor is designed to run workloads that general CPUs eschew. As such, it outperforms the other architectures in the server and bit-level applications. Its lower performance in the desktop integer class (a factor of two worse than the P3) is because the applications in this class are run on a single tile of Raw—these numbers should improve as its ILP compiler matures and is able to schedule integer programs over multiple tiles. Note that for the server benchmarks, we use twenty four independent instantiations of the application, so multithreaded processors can potentially exploit the process-level parallelism.[4]

The VersaGraphs for the five architectures are shown in Figure 5. Each bar in the VersaGraph represents the ratio of the execution time of the corresponding application relative to the highest-performance machine (HPM) for the same application. In other words, each bar represents the ratio R_i/A_i in the range $[0,1]$, where the upper extreme indicates the architecture is the HPM for the application. For the bit-level applications, we synthesized Verilog implementations of two bit-level applications (`80211a` and `8b10b`) using the IBM SA-27E process [41]. This yields ASICs that deliver the best performance for the two applications. At the time of writing this paper, we have yet to generate an ASIC implementation for the `des` benchmark; we expect to complete the synthesis soon. Consequently, the HPM for `des` is the Athlon.

6. RELATED WORK

Benchmarks have played a significant role in the design and optimization of processors. However, previous benchmark suites have all focused on a given class of applications. SPEC, for example, focuses mostly on workstation workloads, and

[4] We will include results from a commercial hyperthreaded Pentium 4 in the final version of this paper.

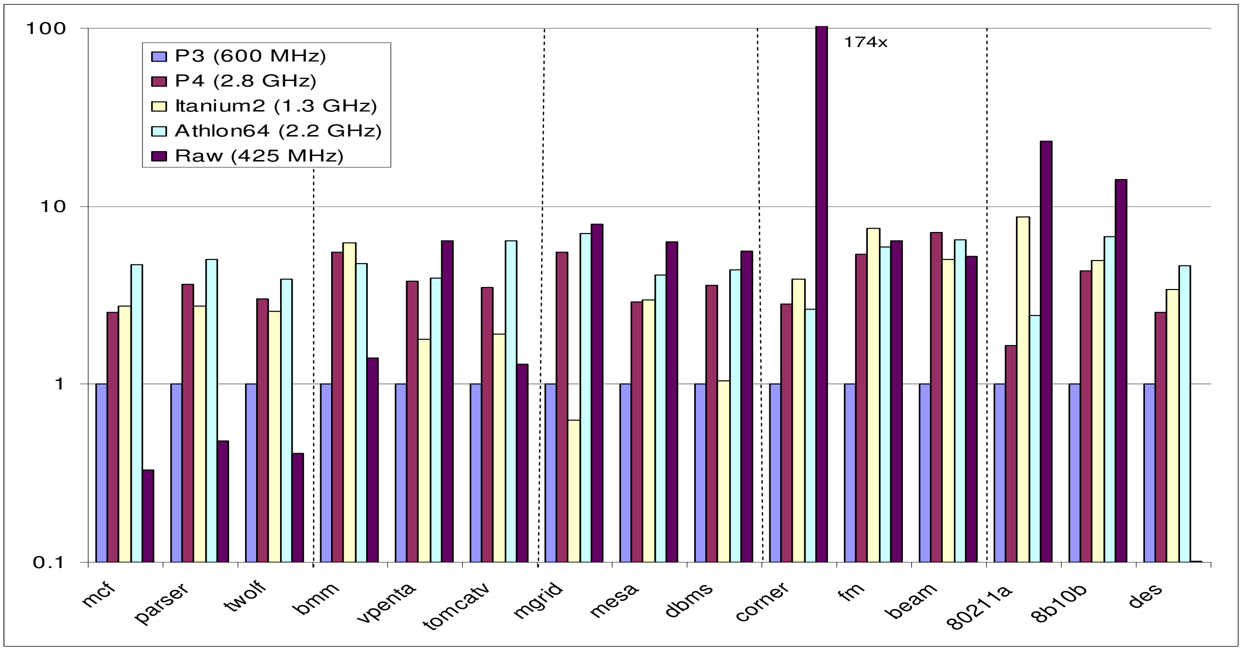


Figure 4. Speedups (in time) relative to the P3 (note a log-scale is used).

Winstone [42] and SYSmark [34] on desktop workloads. MediaBench [16] showed that embedded media applications differ significantly from SPEC in terms of the cache performance, and embedded-systems engineers can reach incorrect conclusions with respect to architectural requirements (if they used improper workloads to calibrate the performance of their media processors). Other benchmark suites such as MiBench [10] have generally focused on embedded systems, NetBench [23], which is often used for network processors, Perfect Club [5] and STREAM [21] for supercomputers, [4] for reconfigurable computing, Splash [43] for multiprocessors, etc. By and large, current benchmark collections focus on a single class of applications. VersaBench, however, attempts to focus on many different application classes.

In selecting the VersaBench constituents, we followed a fundamental approach, focusing on intuitive properties that are easily measurable. For example, we quantify parallelism or IPC, instruction temporal locality, and data locality—both temporal and spatial. This approach is in contrast to previous methodologies that focused on secondary statistics, including branch prediction accuracy, instruction miss rates, and data miss rates. Our methodology for quantifying each of the basis properties is founded upon simple mathematical definitions.

There is a large body of work (e.g., [2, 3, 22, 26, 32]) on measuring temporal and spatial locality, and on average, the definitions of locality are often tuned to a specific context. We share a single characteristic with previous work: the use of a fixed window size (i.e., $\mathcal{W}$ or the number of unique references in a trace segment). One might argue that a variable window size is necessary, as evidenced by the substantial interest in program phases analysis (e.g., [30]) that identifies logical execution units in a program (i.e., the conceptual equivalent of a trace in a trace cache [24, 28]). The approach is intriguing, and we did in fact experiment with dynamically tuned $\mathcal{W}$. We found that in the absence of a fixed segment size, it is difficult to make comparisons and standardize our results.

The Versatility metric, our measure of architecture flexibility, is inspired by the SPECmark metric, and differs only in how speedups are normalized. This single difference is the key innovation in the Versatility metric. Furthermore, the Versatility metric, via VersaGraphs, affords insight into how close the performance of an architecture is to that of an IVM, or concretely, how close the performance of an architecture is to the best performing machine for any given application.

Figure 5. VersaGraphs for five different architectures: Athlon 64 (upper left), Raw (upper right), Itanium 2 (middle left), P4 (middle right) and the P3 (bottom).

7. CONCLUDING REMARKS

We are at the verge of a paradigm shift in the design of microprocessors. Largely spurred by the exponentially increasing chip-level resources and the physical limitations facing contemporary high-performance processors with respect to power and wire-delays, the focus is shifting toward architectures that are more flexible and can exploit computation patterns other than ILP. Notably, multithreading [38] (which is an important part of several new commodity processors), dataflow graph execution [17, 29, 33], and variants of stream, vector and systolic processing [6, 12, 14, 39] are all at the heart of several academic research projects.

Because benchmarks heavily influence architecture innovation and design, there is a need for a new set of applications to represent the broad class of workloads that future flexible architectures are expected to run. The suite, at the very least, must span desktop integer and floating point workloads, server applications, and embedded streaming and bit-level processing. In terms of benchmarks influencing processor designs, one has to look no further than the very well-known SPEC suite, although many other examples exist. To a large extent, previous work has focused on application within a specific context. As processors continue to evolve, there will be an increasing demand for a single benchmark suite that covers a spectrum of applications. As such, this paper proposes VersaBench, a benchmark suite of fifteen benchmarks—a few drawn from other benchmark suites—that span five important computing tiers–namely, desktop integer, desktop floating-point and scientific, server and multithreading, embedded streaming, and embedded bit-level. In selecting the VersaBench constituents, we followed a pragmatic and systematic approach, focusing on five intuitive properties: *predominant data type*, *parallelism*, *instruction temporal locality* (inverse of the control complexity), *data temporal locality*, and *data spatial locality*. Each of the properties can be measured using a simple scalar metric, and we detailed a methodology for doing so. Further, our results demonstrate that each of the benchmark-categories is distinguishable. The VersaBench suite is freely distributed on a website, and we encourage the members of the community to use it to evaluate their respective architectures.

In addition to the collection of benchmarks, this paper also defines Versatility as a single scalar metric to quantify the flexibility of an architecture. Versatility is defined as the geometric mean of the speedups of an architecture for each of the applications in the VersaBench suite, where the speedup of each application is computed relative to highest-performing architecture for that application, in the 2004 time frame from known results at the time of this writing. The Versatility metric mirrors the well-known SPECmark metric, and differs only in the normalizations of the speedups. SPECmarks are normalized relative to a single machine. By contrast, the Versatility metric does not compare the performance of an architecture relative to a single machine, but to the best performing architectures in a given timeframe. As such, Versatility affords engineers the ability to quantify, in absolute terms, the flexibility of their architectures. And further, just as the SPEC metric has influenced architectural innovation in the workstation arena for more than a decade, Versatility is a useful measure to identify opportunities where further innovation will lead to maximal rewards. This paper evaluates the flexibility of several extant processors, as well as a research tiled-processor architecture. We are actively measuring and cataloging the versatility of contemporary processors, and we encourage the community to submit results for their architectures. The results are published as part of the VersaBench website and are publicly available online.

ACKNOWLEDGMENT

The idea of plotting data in the form of VersaGraphs for polymorphic computing architectures first came up in discussions among Janice McMahon, Bob Graybill and one of the authors. This research is funded by DARPA and NSF.

REFERENCES

[1] C. Acosta, S. Vajapeyam, A. Ramirez, and M. Valero. Cde: A compiler-driven, dependence-centric, eager-executing architecture for the billion transistors era. In *Workshop on Complexity-Effective Design (WCED'03)*, June 2003.

[2] A. Agarwal and A. Gupta. *Temporal, Processor, and Spatial Locality in Multiprocessor Memory References*. Plenum Press, 1989. Stu Tewksbury Ed. Also appears as MIT VLSI Memo No. 89-519, 1989.

[3] A. Agarwal, M. Horowitz, and J. Hennessy. An Analytical Cache Model. *ACM Transactions on Computer Systems*, 7(2):184–215, May 1989.

[4] J. Babb, M. Frank, V. Lee, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni, and A. Agarwal. The raw benchmark suite: Computation structures for general purpose computing. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 1997.

[5] M. D. Berry, D. Chen, P. Koss, and D. Kuck. The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers. Technical Report CSRD Report No. 827, CSRD, U. Illinois at Urbana, Nov. 1988.

[6] S. Borkar, R. Cohn, G. Cox, T. Gross, H. T. Kung, M. Lam, M. Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb. Supporting Systolic and Memory Communication in iWarp. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 70–81, June 1990.

[7] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, , and P. Hanrahan. Brook for gpus: Stream computing on graphics hardware. In *SIGGRAPH*, 2004.

[8] S. P. E. Corporation. The SPEC benchmark suites. *http://www.spec.org/*.

[9] L. J. Flynn. Intel halts development of 2 new microprocessors. In *The New York Times*, May 2004.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[11] W. Hwu, S. Mahlke, W. Chen, P. Chang, N. W. r, R. Bringmann, R. Ouellette, R. Hank, T. Kiyohara, G. Haab, J. Holm, and D. Lavery. The superblock: An effective technique for VLIW and super scalar compilation. *Journal of Supercomputing*, Jan. 1993.

[12] U. J. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany. The Imagine Stream Processor. In *Proceedings of ICCD*, 2002.

[13] A. KleinOsowski and D. J. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. In *Computer Architecture Letters*, June 2002.

[14] C. E. Kozyrakis and D. Patterson. A New Direction for Computer Architecture Research. *IEEE Computer*, 30(9), Sept. 1997.

[15] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, , and K. Asanovic. The vector-thread architecture. In *Proceedings of the 31st International Symposium on Computer Architecture (ISCA)*, 2004.

[16] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of the 30th Annual International Symposium on Microarchitecture (MICRO-30)*, Research Triangle Park, NC, December 1997. IEEE Computer Society.

[17] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. Amarasinghe. Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine. In *Proceedings of the Eighth ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 46–57, San Jose, CA, Oct. 1998.

[18] D. J. Lilja. *Measuring Computer Performance – A Practitioner's Guide*. Cambridge University Press, New York, 2000.

[19] S. Mahlke, D. Lin, W. Chen, R. Hank, and R. B. n. Effective compiler support for predicated execution using the h yperblock. In *Proceedings of the 25th Annual International Symposium on Microar chitecture*, pages 45–54, Dec. 1992.

[20] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *Proceedings of the 27th International Symposium on Computer Architecture*, pages 161–170, 2000.

[21] J. McCalpin. STREAM: Sustainable Memory Bandwidth in High Perf. Computers. http://www.cs.virginia.edu/stream.

[22] K. S. McKinley and O. Temam. A quantitative analysis of loop nest locality. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 94–104, Cambridge, Massachusetts, October 1–5, 1996.

[23] G. Memik, W. H. Mangione-Smith, and W. Hu. Netbench: A benchmarking suite for network processors. In *ICCAD*, pages 39–, 2001.

[24] S. Patel, D. H. Friendly, and Y. N. Patt. Critical issues regarding the trace cache fetch mechanism. Technical Report CSE-TR-335-97, University of Michigan, May 1997.

[25] http://www.darpa.mil/ipto/programs/pca/index.htm.

[26] V. Phalke and B. Gopinath. An inter-reference gap model for temporal locality in program behavior. In *Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 291–300, 1995.

[27] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *International Symposium on Microarchitecture (MICRO)*, December 2001.

[28] E. Rotenberg, S. Bennett, and J. E. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. In *Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture*, pages 24–35. IEEE Computer Society, 1996.

[29] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. . k Huh, D. Burger, S. W. Keckler, and C. R. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *2003 ISCA*, pages 422–433, June 2003.

[30] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA)*, June 2003.

[31] J. E. Smith. Characterizing Computer Performance with a Single Number. *Communications of the ACM*, 31(10):1202–1206, October 1988.

[32] J. R. Spirn. *Program Behavior: Models and Measurements*. Operating and Programming Systems Series. Elsevier, New York, 1977.

[33] Swanson, Michelson, Schwerin, and Oskin. WaveScalar. In *International Symposium on Microarchitecture*, 2003.

[34] http://www.bapco.com/products/sysmark2004.

[35] M. B. Taylor. Deionizer: A Tool For Capturing And Embedding I/O Calls. Technical Memo, CSAIL/Laboratory for Computer Science, MIT, 2004. http://cag.csail.mit.edu/∼mtaylor/deionizer.html.

[36] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A Language for Streaming Applications. In *Proceedings of the International Conference on Compiler Construction*, 2002.

[37] TRIMARAN: An infrastructure for research in instru ction level parallelism. http://www.trimaran.org.

[38] D. M. Tulsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, June 1995.

[39] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring It All to Software: Raw Machines. *IEEE Computer*, 30(9):86–93, Sept. 1997. Also available as MIT-LCS-TR-709.

[40] D. W. Wall. Limits of instruction-level parallelism. In *Proc. of the 4th Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, pages 176–188, April 1991.

[41] D. Wentzlaf. A quantitative comparison of reconfigurable, tiled, and conventional architectures on bit-level computation. In *FCCM*, 2004.

[42] http://www.veritest.com/benchmarks/bwinstone.

[43] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and . A. Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture (ISCA)*, June 1995.