The PARSEC Benchmark Suite: Characterization and Architectural Implications

Christian Bienia[†], Sanjeev Kumar[‡], Jaswinder Pal Singh[†] and Kai Li[†]
[†] Department of Computer Science, Princeton University [‡] Microprocessor Technology Labs, Intel
cbienia@cs.princeton.edu

ABSTRACT

This paper presents and characterizes the Princeton Application Repository for Shared-Memory Computers (PARSEC), a benchmark suite for studies of Chip-Multiprocessors (CMPs). Previous available benchmarks for multiprocessors have focused on high-performance computing applications and used a limited number of synchronization methods. PARSEC includes emerging applications in recognition, mining and synthesis (RMS) as well as systems applications which mimic large-scale multithreaded commercial programs. Our characterization shows that the benchmark suite covers a wide spectrum of working sets, locality, data sharing, synchronization and off-chip traffic. The benchmark suite has been made available to the public.

Categories and Subject Descriptors

D.0 [**Software**]: [benchmark suite]

General Terms

Performance, Measurement, Experimentation

Keywords

benchmark suite, performance measurement, multithreading, shared-memory computers

1. INTRODUCTION

Benchmarking is the quantitative foundation of computer architecture research. Benchmarks are necessary to experimentally determine the benefits of new designs. However, to be relevant, a benchmark suite needs to satisfy a number of properties. First, the applications in the suite should be written with the target class of machines in mind. This is necessary to ensure that the architectural features being proposed are relevant and not obviated by minor rewrites of the application. Second, the benchmark suite should represent the important applications on the target machines. Third, the workloads in the benchmark suite should be diverse enough to exhibit the range of behavior of the target applications. Finally, it is important that the programs use state-of-art algorithms and techniques.

As time passes, the relevance of a benchmark suite diminishes. This happens not only because machines evolve and change over time but also because new applications, algorithms, and techniques emerge. New benchmark suites become necessary after significant changes in the architectures or applications.

In fact, dramatic changes have occurred both in mainstream processor designs as well as applications in the last few years. The arrival of chip-multiprocessors (CMPs) with ever increasing number of cores has made parallel machines ubiquitous. At the same time, new applications are emerging that not only organize and catalog data on desktops and the Internet but also deliver improved visual experience [9].

These technology shifts have galvanized research in parallel architectures. Such research efforts rely on existing benchmark suites. However, the existing suites [15, 14, 20, 28] suffer from a number of limitations and are not adequate to evaluate future CMPs (Section 2). The lack of good benchmark suites can hamper parallel architecture research as well as reduce its impact.

To address this problem, we created a publicly available benchmark suite called PARSEC in collaboration with Intel Corporation. It includes not only a number of important RMS applications [9] but also several leading-edge applications from Princeton University, Stanford University, and the open-source domain. The goal is to create a suite of emerging workloads that can drive CMP research.

A recent study [6] quantitatively demonstrated that the characteristics of PARSEC are significantly different from SPLASH-2 (one of the most widely used parallel benchmark suites). This suggests that using newer benchmark suites like PARSEC is necessary.

The widely perceived need for such a benchmark suite is proven by the number of researchers who are already downloading and using PARSEC. Within the first 6 months of being made publicly available, the benchmark suite has been downloaded more than 500 times by researchers throughout the world. The first papers using PARSEC have been submitted.

This paper makes three contributions:

- It identifies shortcomings of commonly used benchmark suites and explains why they might be less relevant to evaluate CMPs (Section 2).
- We present and characterize PARSEC, a new benchmark suite for CMPs that is diverse enough in order to allow representative conclusions (Sections 3 - 8).
- Based on our characterization of PARSEC, we analyze what properties future CMPs must have in order to be able to deliver scalable performance for emerging applications (Sections 6 - 8).

2. MOTIVATION

The goal of this work is to define a benchmark suite that can be used to design the next generation of processors. In this section, we first present the requirements for such a suite. We then discuss how the existing benchmarks fail to meet these requirements.

2.1 Requirements for a Benchmark Suite

We have the following five requirements for a benchmark suite:

Multithreaded Applications Shared-memory CMPs are already ubiquitous. The trend for future processors is to deliver large performance improvements through increasing core counts on CMPs while only providing modest serial performance improvements. Consequently, applications that require additional processing power will need to be parallel.

Emerging Workloads Rapidly increasing processing power is enabling a new class of applications whose computational requirements were beyond the capabilities of the earlier generation of processors [9]. Such applications are significantly different from earlier applications (see Section 3). Future processors will be designed to meet the demands of these emerging applications and a benchmark suite should represent them.

Diverse Applications are increasingly diverse, run on a variety of platforms and accommodate different usage models. They include both interactive applications like computer games, offline applications like data mining programs and programs with different parallelization models. Specialized collections of benchmarks can be used to study some of these areas in more detail, but decisions about general-purpose processors should be based on a diverse set of applications.

Employ State-of-Art Techniques A number of application areas have changed dramatically over the last decade and use very different algorithms and techniques. Visual applications for example have started to increasingly integrate physics simulations to generate more realistic animations [13]. A benchmark should not only represent emerging applications but also use state-of-art techniques.

Support Research A benchmark suite intended for research has additional requirements compared to one used for benchmarking real machines alone. Benchmark suites intended for research usually go beyond pure scoring systems and provide infrastructure to instrument, manipulate, and perform detailed simulations of the included programs in an efficient manner.

2.2 Limitations of Existing Benchmark Suites

In the remaining part of this section we analyze how existing benchmark suites fall short of the presented requirements and must thus be considered unsuitable for evaluating CMP performance.

SPLASH-2 SPLASH-2 is a suite composed of multithreaded applications [26] and hence seems to be an ideal candidate to measure performance of CMPs. However, its program collection is skewed towards HPC and graphics programs. It thus does not include parallelization models such as the pipeline model which are used in other application areas. SPLASH-2 should furthermore not be considered state-of-art anymore. `Barnes` for example implements the Barnes-Hut algorithm for N-body simulation [4]. For galaxy simulations it has largely been superseded by the TreeSPH [12] method, which can also account for mass such as dark matter which is not concentrated in bodies. However, even for pure N-body simulation `barnes` must be considered outdated. In 1995 Xu proposed a hybrid algorithm which combines the hierarchical tree algorithm and the Fourier-based Particle-Mesh (PM) method to the superior TreePM method [27]. Our analysis shows that similar issues exist for a number of other applications of the suite including `raytrace` and `radiosity`.

SPEC CPU2006 and OMP2001 SPEC CPU2006 and SPEC OMP2001 are two of the largest and most significant collections of benchmarks. They provide a snapshot of current scientific and engineering applications. Computer architecture research, however, commonly focuses on the near future and should thus also consider emerging applications. Workloads such as systems programs and parallelization models which employ the producer-consumer model are not included. SPEC CPU2006 is furthermore a suite of serial programs that is not intended for studies of parallel machines.

Other Benchmark Suites Besides these major benchmark suites, several smaller workload collections exist. They were usually designed to study a specific program area and are thus limited to a single application domain. Therefore they usually include a smaller set of applications than a diverse benchmark suite typically offers. Due to these limitations they are commonly not used for scientific studies which do not restrict themselves to the covered application domain. Examples for these types of benchmark suites are ALPBench [15], BioParallel [14], MediaBench [18], MineBench [20] and PhysicsBench [28]. Because of their different focus we do not discuss these suites in more detail.

3. THE PARSEC BENCHMARK SUITE

One of the goals of the PARSEC suite was to assemble a program selection that is large and diverse enough to be sufficiently representative for scientific studies. It consists of 9 applications and 3 kernels which were chosen from a wide range of application domains. PARSEC workloads were selected to include different combinations of parallel models, machine requirements and runtime behaviors. All benchmarks are written in C/C++ because of the continuing popularity of these languages in the near future.

PARSEC meets all the requirements outlined in Section 2.1:

- Each of the applications has been parallelized.
- The PARSEC benchmark suite is not skewed towards HPC programs, which are abundant but represent only a niche. It focuses on emerging workloads.
- The workloads are diverse and were chosen from many different areas such as computer vision, media processing, computational finance, enterprise servers and animation physics. PARSEC is more diverse than SPLASH-2 [6].
- Each of the applications chosen represents the state-of-art technique in its area.
- PARSEC supports computer architecture research in a number of ways. The most important one is that for each workload six input sets with different properties are defined (Section 3.1).

The characteristics of the included workloads differ substantially from SPLASH-2 [6]. Recent technology trends such as the emergence of CMPs and the growth of world data seem to have a strong impact on workload behavior.

3.1 Input Sets

PARSEC defines six input sets for each benchmark:

test A very small input set to test the basic functionality of the program.

simdev A very small input set which guarantees basic program behavior similar to the real behavior, intended for simulator test and development.

simsmall, simmedium and simlarge Input sets of different sizes suitable for simulations.

native A large input set intended for native execution.

Program	Problem Size	Instructions (Billions)				Synchronization Primitives		
		Total	FLOPS	Reads	Writes	Locks	Barriers	Conditions
`blackscholes`	65,536 options	2.67	1.14	0.68	0.19	0	8	0
`bodytrack`	4 frames, 4,000 particles	14.03	4.22	3.63	0.95	114,621	619	2,042
`canneal`	400,000 elements	7.33	0.48	1.94	0.89	34	0	0
`dedup`	184 MB data	37.1	0	11.71	3.13	158,979	0	1,619
`facesim`	1 frame, 372,126 tetrahedra	29.90	9.10	10.05	4.29	14,541	0	3,137
`ferret`	256 queries, 34,973 images	23.97	4.51	7.49	1.18	345,778	0	1255
`fluidanimate`	5 frames, 300,000 particles	14.06	2.49	4.80	1.15	17,771,909	0	0
`freqmine`	990,000 transactions	33.45	0.00	11.31	5.24	990,025	0	0
`streamcluster`	16,384 points per block, 1 block	22.12	11.6	9.42	0.06	191	129,600	127
`swaptions`	64 swaptions, 20,000 simulations	14.11	2.62	5.08	1.16	23	0	0
`vips`	1 image, 2662×5500 pixels	31.21	4.79	6.71	1.63	33,586	0	6,361
`x264`	128 frames, 640×360 pixels	32.43	8.76	9.01	3.11	16,767	0	1,056

Table 1: Breakdown of instructions and synchronization primitives for input set `simlarge` on a system with 8 cores. All numbers are totals across all threads. Numbers for synchronization primitives also include primitives in system libraries. "Locks" and "Barriers" are all lock- and barrier-based synchronizations, "Conditions" are all waits on condition variables.

`test` and `simdev` are merely intended for testing and development and should not be used for scientific studies. The three simulator inputs for studies vary in size, but the general trend is that larger input sets contain bigger working sets and more parallelism. Finally, the `native` input set is intended for performance measurements on real machines and exceeds the computational demands which are generally considered feasible for simulation by orders of magnitude. Table 1 shows a breakdown of instructions and synchronization primitives of the `simlarge` input set which we used for the characterization study.

3.2 Workloads

The following workloads are part of the PARSEC suite:

blackscholes This application is an Intel RMS benchmark. It calculates the prices for a portfolio of European options analytically with the Black-Scholes partial differential equation (PDE) [7]. There is no closed-form expression for the Black-Scholes equation and as such it must be computed numerically.

bodytrack This computer vision application is an Intel RMS workload which tracks a human body with multiple cameras through an image sequence [8]. This benchmark was included due to the increasing significance of computer vision algorithms in areas such as video surveillance, character animation and computer interfaces.

canneal This kernel was developed by Princeton University. It uses cache-aware simulated annealing (SA) to minimize the routing cost of a chip design [3]. `Canneal` uses fine-grained parallelism with a lock-free algorithm and a very aggressive synchronization strategy that is based on data race recovery instead of avoidance.

dedup This kernel was developed by Princeton University. It compresses a data stream with a combination of global and local compression that is called 'deduplication'. The kernel uses a pipelined programming model to mimic real-world implementations. The reason for the inclusion of this kernel is that deduplication has become a mainstream method for new-generation backup storage systems [23].

facesim This Intel RMS application was originally developed by Stanford University. It computes a visually realistic animation of the modeled face by simulating the underlying physics [24]. The workload was included in the benchmark suite because an increasing number of animations employ physical simulation to create more realistic effects.

ferret This application is based on the Ferret toolkit which is used for content-based similarity search [16]. It was developed by Princeton University. The reason for the inclusion in the benchmark suite is that it represents emerging next-generation search engines for non-text document data types. In the benchmark, we have configured the Ferret toolkit for image similarity search. Ferret is parallelized using the pipeline model.

fluidanimate This Intel RMS application uses an extension of the Smoothed Particle Hydrodynamics (SPH) method to simulate an incompressible fluid for interactive animation purposes [19]. It was included in the PARSEC benchmark suite because of the increasing significance of physics simulations for animations.

freqmine This application employs an array-based version of the FP-growth (Frequent Pattern-growth) method [10] for Frequent Itemset Mining (FIMI). It is an Intel RMS benchmark which was originally developed by Concordia University. `freqmine` was included in the PARSEC benchmark suite because of the increasing use of data mining techniques.

streamcluster This RMS kernel was developed by Princeton University and solves the online clustering problem [21]. `streamcluster` was included in the PARSEC benchmark suite because of the importance of data mining algorithms and the prevalence of problems with streaming characteristics.

swaptions The application is an Intel RMS workload which uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of swaptions [11]. `Swaptions` employs Monte Carlo (MC) simulation to compute the prices.

vips This application is based on the VASARI Image Processing System (VIPS) [17] which was originally developed through several projects funded by European Union (EU) grants. The benchmark version is derived from a print on demand service that is offered at the National Gallery of London, which is also the current maintainer of the system. The benchmark includes fundamental image operations such as an affine transformation and a convolution.

x264 This application is an H.264/AVC (Advanced Video Coding) video encoder. H.264 describes the lossy compression of a video stream [25] and is also part of ISO/IEC MPEG-4. The flexibility and wide range of application of the H.264 standard and its ubiquity in next-generation video systems are the reasons for the inclusion of `x264` in the PARSEC benchmark suite.

4. METHODOLOGY

In this section we explain how we characterized the PARSEC benchmark suite. We are interested in the following characteristics:

Parallelization PARSEC benchmarks use different parallel models which have to be analyzed in order to know whether the programs can scale well enough for the analysis of CMPs of a certain size.

Working sets and locality Knowledge of the cache requirements of a workload are necessary to identify benchmarks suitable for the study of CMP memory hierarchies.

Communication-to-computation ratio and sharing The communication patterns of a program determine the potential impact of private caches and the on-chip network on performance.

Off-chip traffic The off-chip traffic requirements of a program are important to understand how off-chip bandwidth limitations of a CMP can affect performance.

In order to characterize all applications, we had to make several trade-off decisions. Given a limited amount of computational resources, higher accuracy comes at the expense of a lower number of experiments. We followed the approach of similar studies [26, 14] and chose faster but less accurate execution-driven simulation to characterize the PARSEC workloads. This approach is feasible because we limit ourselves to study fundamental program properties which should have a high degree of independence from architectural details. Where possible we supply measurement results from real machines. This methodology allowed us to gather the large amount of data which we present in this study. We preferred machine models comparable to real processors over unrealistic models which might have been a better match for the program needs.

4.1 Experimental Setup

We used CMP$im [14] for our workload characterization. CMP-$im is a plug-in for Pin [22] that simulates the cache hierarchy of a CMP. Pin is similar to the ATOM toolkit for Compaq's Tru64 Unix on Alpha processors. It uses dynamic binary instrumentation to insert routines at arbitrary points in the instruction stream. For the characterization we simulate a single-level cache hierarchy of a CMP and vary its parameters. The baseline cache configuration was a shared 4-way associative cache with 4 MB capacity and 64 byte lines. By default the workloads used 8 cores. All experiments were conducted on a set of Symmetric Multiprocessor (SMP) machines with x86 processors and Linux. The programs were compiled with `gcc` 4.2.1.

Because of the large computational cost we could not perform simulations with the `native` input set, instead we used the `simlarge` inputs for all simulations and analytically describe any differences between the two sets of which we know.

4.2 Methodological Limitations and Error Margins

For their characterization of the SPLASH-2 benchmark suite, Woo et al. fixed a timing model which they used for all experiments [26]. They give two reasons: First, nondeterministic programs would otherwise be difficult to compare because different execution paths could be taken, and second, the characteristics they study are largely independent from an architecture. They also state that they believe that the timing model should have only a small impact on the results. While we use similar characteristics and share this belief, we think a characterization study of multithreaded programs should nevertheless analyze the impact of nondeterminism on the reported data. Furthermore, because our methodology is based on execution on real machines combined with dynamic binary instrumentation, it can introduce additional latencies, and a potential concern is that the nondeterministic thread schedule is altered in a way that might affect our results in unpredictable ways. We therefore conducted a sensitivity analysis to quantify the impact of nondeterminism.

Alameldeen and Wood studied the variability of nondeterministic programs in more detail and showed that even small pseudo-random perturbations of memory latencies are effective to force alternate execution paths [2]. We adopted their approach and modified CMP$im to add extra delays to its analysis functions. Because running all experiments multiple times as Alameldeen and Wood did would be prohibitively expensive, we instead decided to randomly select a subset of all experiments for each metric which we use and report its error margins.

The measured quantities deviated by no more than $\pm 0.04\%$ from the average, with the following two exceptions. The first excpetion is metrics of data sharing. In two cases (`bodytrack` and `swaptions`) the classification is noticeably affected by the nondeterminism of the program. This is partially caused because shared and thread-private data contend aggressively for a limited amount of cache capacity. The high frequency of evictions made it difficult to classify lines and accesses as shared or private. In these cases, the maximum deviation of the number of accesses from the average was as high as $\pm 4.71\%$, and the amount of sharing deviated by as much as $\pm 15.22\%$. We considered this uncertainty in our study and did not draw any conclusions where the variation of the measurements did not allow it. The second case of high variability is when the value of the measured quantity is very low (below 0.1% miss rate or corresponding ratio). In these cases the nondeterministic noise made measurements difficult. We do not consider this a problem because in this study we focus on trends of ratios, and quantities that small do not have a noticeable impact. It is however an issue for the analysis of working sets if the miss rate falls below this threshold and continues to decrease slowly. Only few programs are affected, and our estimate of their working set sizes might be slightly off in these cases. This is primarily an issue inherent to experimental working set analysis, since it requires well-defined points of inflection for conclusive results. Moreover, we believe that in these cases the working set size varies nondeterministically, and researchers should expect slight variations for each benchmark run.

The implications of these results are twofold: First, they show that our methodology is not susceptible to the nondeterministic effects of multithreaded programs in a way that might invalidate our findings. Second, they also confirm that the metrics which we present in this paper are fundamental program properties which cannot be distorted easily. The reported application characteristics are likely to be preserved on a large range of architectures.

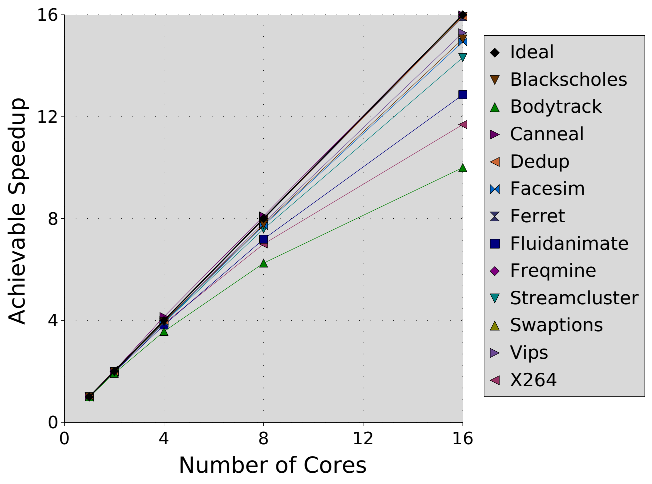

Figure 1: Upper bound for speedup of PARSEC workloads based on instruction count.

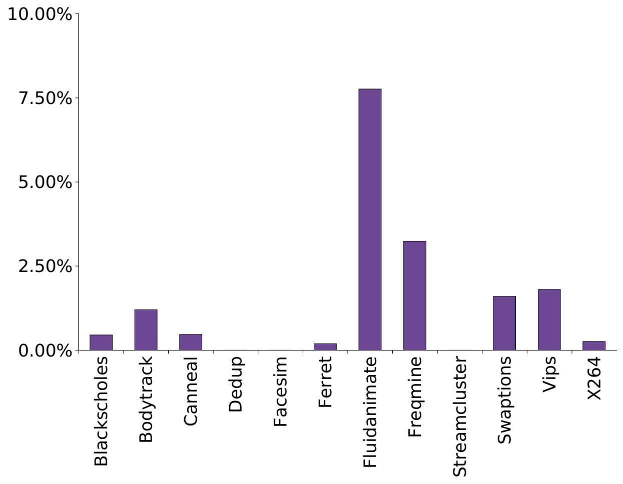

Figure 2: Parallelization overhead of PARSEC benchmarks. The chart shows the slowdown of the parallel version on 1 core over the serial version.

5. PARALLELIZATION

In this section we discuss the parallelization of the PARSEC suite. As we will see in Section 6, several PARSEC benchmarks (`canneal`, `dedup`, `ferret` and `freqmine`) have working sets so large they should be considered unbounded for an analysis. These working sets are only limited by the amount of main memory in practice and they are actively used for inter-thread communication. The inability to use caches efficiently is a fundamental property of these program and affects their concurrent behavior. Furthermore, `dedup` and `ferret` use a complex, heterogeneous parallelization model in which specialized threads execute different functions with different characteristics at the same time. These programs employ a pipeline with dedicated thread pools for each parallelized pipeline stage. Each thread pool has enough threads to occupy the whole CMP, and it is the responsibility of the scheduler to assign cores to threads in a manner that maximizes the overall throughput of the pipeline. Over time, the number of threads active for each stage will converge against the inverse throughput ratios of the individual pipeline stages relative to each other.

Woo et al. use an abstract machine model with a uniform instruction latency of one cycle to measure the speedups of the SPLASH-2 programs [26]. They justify their approach by pointing out that the impact of the timing model on the characteristics which they measure - including speedup - is likely to be low. Unfortunately, this is not true in general for PARSEC workloads. While we have verified in Section 4.2 that the fundamental program properties such as miss rate and instruction count are largely not susceptible to timing shocks, the synchronization and timing behavior of the programs is. Using a timing model with perfect caches significantly alters the behavior of programs with unbounded working sets, for example how long locks to large, shared data structures are held. Moreover, any changes of the timing model have a strong impact on the number of active threads of programs which employ thread specialization. It will thus affect the load balance and synchronization behavior of these workloads. We believe it is not possible to discuss the timing behavior of these programs without also considering for example different schedulers, which is beyond the scope of this paper. Similar dependencies of commercial workloads on their environment are already known [5, 1].

Unlike Woo et al. who measured *actual* concurrency on an abstract machine, we therefore decided to analyze *inherent* concurrency and its limitations. Our approach is based on the number of executed instructions in parallel and serial regions of the code. We neglect any delays due to blocking on contended locks and load imbalance. This methodology is feasible because we do not study performance, our interest is in fundamental program characteristics. The presented data is largely timing-independent and a suitable measure of the concurrency inherent in a workload. The results in Figure 1 show the maximum achievable speedup measured that way. The numbers account for limitations such as unparallelized code sections, synchronization overhead and redundant computations. PARSEC workloads can achieve *actual* speedups close to the presented numbers. We verified on a large range of architectures that lock contention and other timing-dependent factors are not limiting factors, but we know of no way to show it in a platform-independent way given the complications outlined above. The maximum speedup of `bodytrack`, `x264` and `streamcluster` is limited by serial sections of the code. `fluidanimate` is primarily limited by growing parallelization overhead. On real machines, `x264` is furthermore bound by a data dependency between threads, however this has only a noticeable impact on machines larger than the ones described here. It is recommended to run `x264` with more threads than cores, since modeling and exposing these dependencies to the scheduler is a fundamental aspect of its parallel algorithm, comparable to the parallel algorithms of `dedup` and `ferret`. Figure 2 shows the slowdown of the parallel version on 1 core over the serial version. The numbers show that all workloads use efficient parallel algorithms which are not substantially slower than the corresponding serial algorithms.

PARSEC programs scale well enough to study CMPs. We believe they are also useful on machines larger than the ones analyzed here. The PARSEC suite exhibits a wider variety of parallelization models than previous benchmark suites such as the pipeline model. Some of its workloads can adapt to different timing models and can use threads to hide latencies. It is important to analyze these programs in the context of the whole system.

6. WORKING SETS AND LOCALITY

The temporal locality of a program can be estimated by analyzing how the miss rate of a processor's cache changes as its capacity is varied. Often the miss rate does not decrease continuously as the size of a cache is increased, but stays on a certain level and then

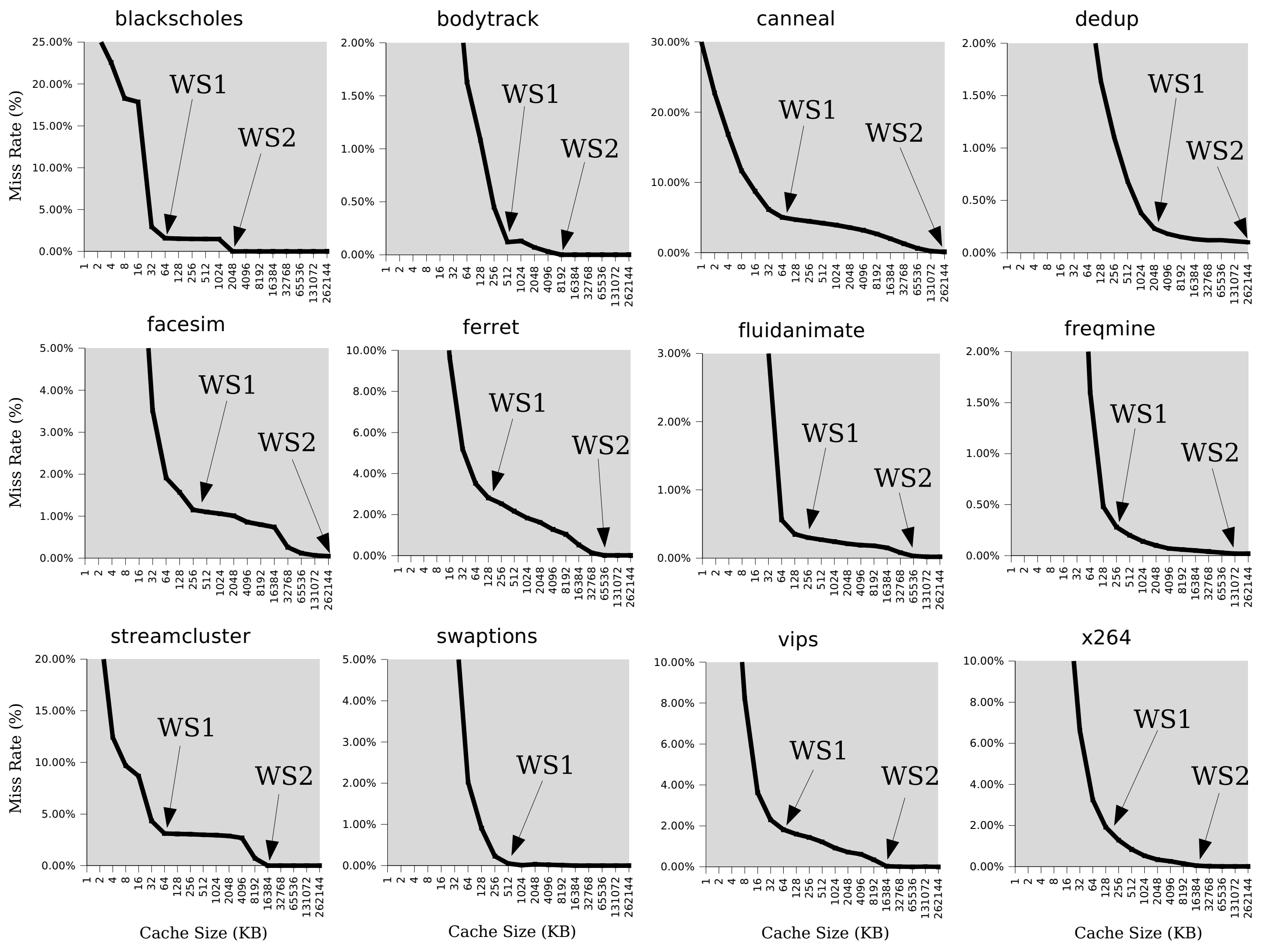

Figure 3: Miss rates versus cache size. Data assumes a shared 4-way associative cache with 64 byte lines. WS1 and WS2 refer to important working sets which we analyze in more detail in Table 2. Cache requirements of PARSEC benchmark programs can reach hundreds of megabytes.

Program	**Input Set** `simlarge`						**Input Set** `native`
	Working Set 1			**Working Set 2**			**Working Set 2**
	Data Structure(s)	**Size**	**Growth Rate**	**Data Structure(s)**	**Size**	**Growth Rate**	**Size Estimate**
`blackscholes`	options	64 KB	C	portfolio data	2 MB	C	same
`bodytrack`	edge maps	512 KB	const.	input frames	8 MB	const.	same
`canneal`	elements	64 KB	C	netlist	256 MB	DS	2 GB
`dedup`	data chunks	2 MB	C	hash table	256 MB	DS	2 GB
`facesim`	tetrahedra	256 KB	C	face mesh	256 MB	DS	same
`ferret`	images	128 KB	C	data base	64 MB	DS	128 MB
`fluidanimate`	cells	128 KB	C	particle data	64 MB	DS	128 MB
`freqmine`	transactions	256 KB	C	FP-tree	128 MB	DS	1 GB
`streamcluster`	data points	64 KB	C	data block	16 MB	user-def.	256 MB
`swaptions`	swaptions	512 KB	C	same as WS1	same	same	same
`vips`	image data	64 KB	C	image data	16 MB	C	same
`x264`	macroblocks	128 KB	C	reference frames	16 MB	C	same

Table 2: Important working sets and their growth rates. DS represents the data set size and C is the number of cores. Working set sizes are taken from Figure 3. Values for `native` input set are analytically derived estimates. Working sets that grow proportional to the number of cores C are aggregated private working sets and can be split up to fit into correspondingly smaller, private caches.

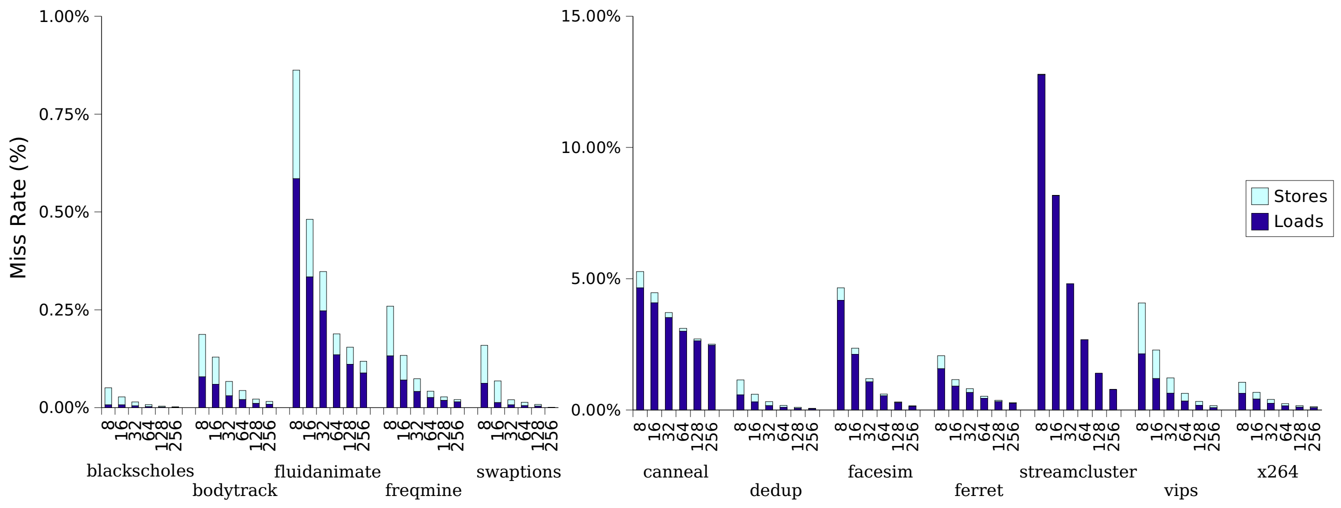

Figure 4: Miss rates as a function of line size. Data assumes 8 cores sharing a 4-way associative cache with 4 MB capacity. Miss rates are broken down to show the effect of loads and stores.

makes a sudden jump to a lower level when the capacity becomes large enough to hold the next important data structure. For CMPs an efficient functioning of the last cache level on the chip is crucial because a miss in the last level will require an access to off-chip memory.

To analyze the working sets of the PARSEC workloads we studied a cache shared by all processors. Our results are presented in Figure 3. In Table 2 we summarize the important characteristics of the identified working sets. Most workloads exhibit well-defined working sets with clearly identifiable points of inflection. Compared to SPLASH-2, PARSEC working sets are significantly larger and can reach hundreds of megabytes such as in the cases of `canneal` and `freqmine`.

Two types of workloads can be distinguished: The first group contains benchmarks such as `bodytrack` and `swaptions` which have working sets no larger than 16 MB. These workloads have a limited need for caches with a bigger capacity, and the latest generation of CMPs often already has caches sufficiently large to accommodate most of their working sets. The second group of workloads is composed of the benchmarks `canneal`, `ferret`, `facesim`, `fluidanimate` and `freqmine`. These programs have very large working sets of sizes 65 MB and more, and even with a relatively constrained input set such as `simlarge`, their working sets can reach hundreds of megabytes. Moreover, the need of those workloads for cache capacity is nearly insatiable and grows with the amount of data which they process. In Table 2 we give our estimates for the largest working set of each PARSEC workload for the `native` input set. In several cases they are significantly larger and can even reach gigabytes. These large working sets are often the consequence of an algorithm that operates on large amounts of collected input data. `ferret` for example keeps a data base of feature vectors of images in memory to find the images most similar to a given query image. The cache and memory needs of these applications should be considered unbounded, as they become more useful to their users if they can work with increased amounts of data. Programs with unbounded working sets are `canneal`, `dedup`, `ferret` and `freqmine`.

In Figure 4 we present our analysis of the spatial locality of the PARSEC workloads. The data shows how the miss rate of a shared cache changes with line size. All programs benefit from larger cache lines, but to different extents. `facesim`, `fluidanimate` and `streamcluster` show the greatest improvement as the line size is increased, up to the the maximum value of 256 bytes which we used. These programs have streaming behavior, and an increased line size has a prefetching effect which these workloads can take advantage of. `facesim` for example spends most of its time updating the position-based state of the model, for which it employs an iterative Newton-Raphson algorithm. The algorithm iterates over the elements of a sparse matrix which is stored in two one-dimensional arrays, resulting in a streaming behavior. All other programs also show good improvement of the miss rate with larger cache lines, but only up to line sizes of about 128 bytes. The miss rate is not substantially reduced with larger lines. This is due to a limited size of the basic data structures employed by the programs. They represent independent logical units, each of which is intensely worked with during a computational phase. For example, `x264` operates on macroblocks of 8×8 pixels at a time, which limits the sizes of the used data structures. Processing a macroblock is computationally intensive and largely independent from other macroblocks. Consequently, the amount of spatial locality is bounded in these cases.

For the rest of our analysis we chose a cache capacity of 4 MB for all experiments. We could have used a matching cache size for each workload, but that would have made comparisons very difficult, and the use of very small or very large cache sizes is not realistic. Moreover, in the case of the workloads with an unbounded working set size, a working set which completely fits into a cache would be an artifact of the limited simulation input size and would not reflect realistic program behavior.

7. COMMUNICATION-TO-COMPUTATION RATIO AND SHARING

In this section we discuss how PARSEC workloads use caches to communicate. Most PARSEC benchmarks share data intensely. Two degrees of sharing can be distinguished: Shared data can be read-only during the parallel phase, in which case it is only used for lookups and analysis. Input data is frequently used in such a way. But shared data can also be used for communication between threads, in which case it is also modified during the parallel phase. In Figure 5 we show how the line size affects sharing. The data combines the effects of false sharing and the access pattern of the program due to constrained cache capacity. In Figure 6, we ana-

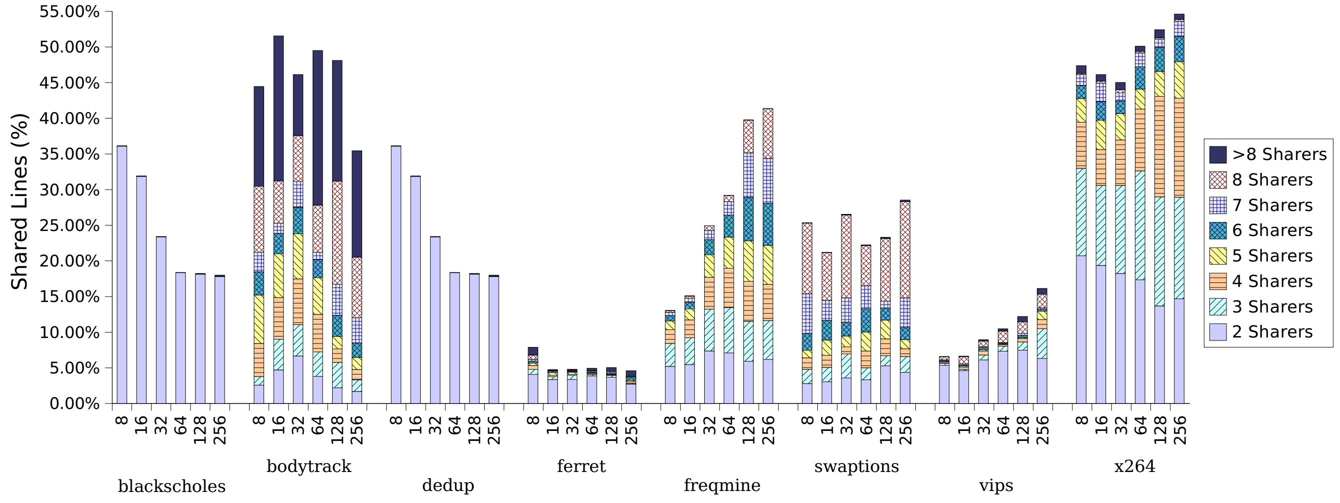

Figure 5: Portion of a 4-way associative cache with 4 MB capacity which is shared by 8 cores. The line size is varied from 8 to 256 bytes.

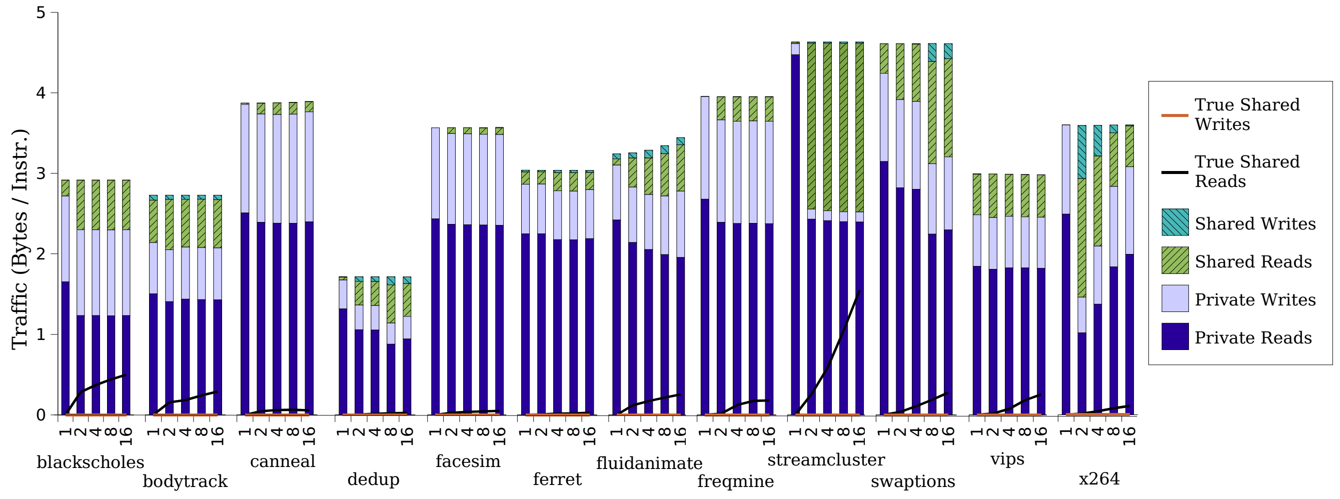

Figure 6: Traffic from cache in bytes per instruction for 1 to 16 cores. Data assumes a shared 4-way associative cache with 64 byte lines.

lyze how the program uses its data. The chart shows what data is accessed and how intensely it is used. The information is broken down in two orthogonal ways, resulting in four possible types of accesses: Read and write accesses and accesses to thread-private and shared data. Additionally, we give numbers for true shared accesses. An access is a true access if the last reference to that line came from another thread. True sharing does not count repeated accesses by the same thread. It is a useful metric to estimate the requirements for the cache coherence mechanism of a CMP: A true shared write can trigger a coherence invalidate or update, and a true shared read might require the replication of data. All programs exhibit very few true shared writes.

Four programs (`canneal`, `facesim`, `fluidanimate` and `streamcluster`) showed only trivial amounts of sharing. They have therefore not been included in Figure 5. In the case of `canneal`, this is a result of the small cache capacity. Most of its large working set is shared and actively worked with by all threads. However, only a minuscule fraction of it fits into the cache, and the probability that a line is accessed by more than one thread before it gets replaced is very small in practice. With a 256 MB cache, 58% of its cached data is shared. `blackscholes` shows a substantial amount of sharing, but almost all its shared data is only accessed by two threads. This is a side-effect of the parallelization model: At the beginning of the program, the boss threads initializes the portfolio data before it spawns worker threads which process parts of it in a data-parallel way. As such, the entire portfolio is shared between the boss thread and its workers, but the worker threads can process the options independently from each other and do not have to communicate with each other. `ferret` shows a modest amount of data sharing. Like the sharing behavior of `canneal`, this is caused by severely constrained cache capacity. `ferret` uses a database that is scanned by all threads to find entries similar to the query image. However, the size of the database is practically unbounded, and because threads do not coordinate their scans with each other it is unlikely that a cache line gets accessed more than once. `bodytrack` and `freqmine` exhibit substantial amounts of sharing due to the fact that threads process the same data. The strong increase of sharing of `freqmine` is caused by false sharing, as the program uses an

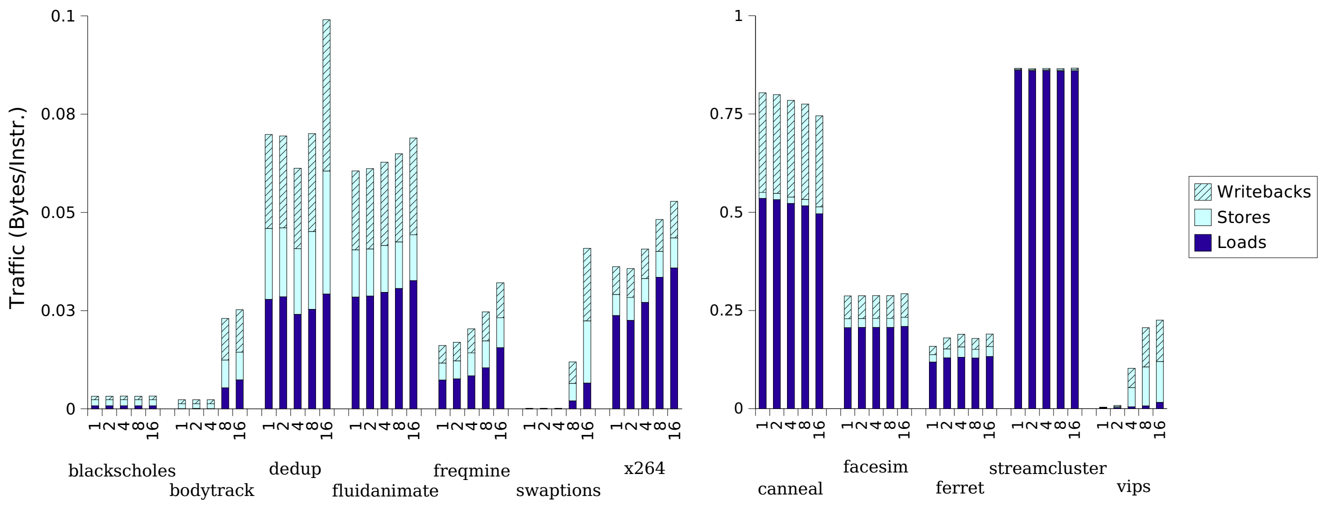

Figure 7: Breakdown of off-chip traffic for 1 to 16 cores. Data assumes a 4-way associative 4 MB cache with 64 byte lines, allocate-on-store and write-back policy.

array-based tree as its main data structure. Larger cache lines will contain more nodes, increasing the chance that the line is accessed by multiple threads. `vips` has some shared data which is mostly used by only two threads. This is also predominantly an effect of false sharing since image data is stored in a consecutive array which is processed in a data-parallel way by threads. `x264` uses significant amounts of shared data, most of which is only accessed by a low number of threads. This data is the reference frames, since a thread needs this information from other stages in order to encode the frame it was assigned. Similarly, the large amount of shared data of `dedup` is the input which is passed from stage to stage.

Most PARSEC workloads use a significant amount of communication, and in many cases the volume of traffic between threads can be so high that efficient data exchange via a shared cache is severely constrained by its capacity. An example for this is `x264`. Figure 6 shows a large amount of writes to shared data, but contrary to intuition its share diminishes rapidly as the number of cores is increased. This effect is caused by a growth of the working sets of `x264`: Table2 shows that both working set WS1 and WS2 grow proportional to the number of cores. WS1 is mostly composed of thread-private data and is the one which is used more intensely. WS2 contains the reference frames and is used for inter-thread communication. As WS1 grows, it starts to displace WS2, and the threads are forced to communicate via main memory. Two more programs which communicate intensely are `dedup` and `ferret`. Both programs use the pipeline parallelization model with dedicated thread pools for each parallel stage, and all data has to be passed from stage to stage. `fluidanimate` also shows a large amount of inter-thread communication, and its communication needs grow as the number of threads increase. This is caused by the spatial partitioning that `fluidanimate` uses to distribute the work to threads. Smaller partitions mean a worse surface to volume ratio, and communication grows with the surface.

Overall, most PARSEC workloads have complex sharing patterns and communicate actively. Pipelined programs can require a large amount of bandwidth between cores in order to communicate efficiently. Shared caches with insufficient capacity can limit the communication efficiency of workloads, since shared data structures might get displaced to memory.

8. OFF-CHIP TRAFFIC

In this section we analyze what the off-chip bandwidth requirements of PARSEC workloads are. Our goal is to understand how the traffic of an application grows as the number of cores of a CMP increases and how the memory wall will limit performance. We again simulated a shared cache and analyze how traffic develops as the number of cores increases. Our results are presented in Figure 7.

The data shows that the off-chip bandwidth requirements of the `blackscholes` workload are small enough so that memory bandwidth is unlikely to be an issue. `bodytrack`, `dedup`, `fluidanimate`, `freqmine`, `swaptions` and `x264` are more demanding. Moreover, these programs exhibit a growing bandwidth demand per instruction as the number of cores increases. In the case of `bodytrack`, most off-chip traffic happens in short, intense bursts since the off-chip communication predominantly takes place during the edge map computation. This phase is only a small part of the serial runtime, but on machines with constrained memory bandwidth it quickly becomes the limiting factor for scalability. The last group of programs is composed of `canneal`, `facesim`, `ferret`, `streamcluster` and `vips`. These programs have very high bandwidth requirements and also large working sets. `canneal` shows a decreasing demand for data per instruction with more cores. This behavior is caused by improved data sharing.

It is important to point out that these numbers do not take the increasing instruction throughput of a CMP into account as its number of cores grows. A constant traffic amount in Figure 7 means that the bandwidth requirements of an application which scales linearly will grow exponentially. Since many PARSEC workloads have high bandwidth requirements and working sets which exceed conventional caches by far, off-chip bandwidth will be their most severe limitation of performance. Substantial architectural improvements are necessary to allow emerging workloads to take full advantage of larger CMPs.

9. FUTURE WORK

Page limitations forced us to restrict the scope of our study to workload characterization using only one of the available input sets. Additional work is necessary to establish PARSEC as a mature benchmark suite. The other available input sets should also be analyzed and compared for similarity. PARSEC already is a large improvement if age, domains of included applications and covered parallelization models alone are considered. Any weaknesses in the spectrum of covered programs which might still exist could be identified with a coverage analysis. Furthermore, the understanding of workloads could be further improved if the individual kernels and phases of the benchmarks are analyzed independently from each other.

10. CONCLUSIONS

The PARSEC benchmark suite is designed to provide parallel programs for the study for CMPs. PARSEC can be used to drive research efforts by application demands. It focuses on emerging desktop and server applications and does not have the limitations of other benchmark suites. It is diverse enough to be considered representative, it is not skewed towards HPC programs, it uses state-of-art algorithms and it supports research. In this study we characterized the PARSEC workloads to provide the basic understanding necessary to allow other researchers the effective use of PARSEC for their studies. We analyzed the parallelization, the working sets and locality, the communication-to-computation ratio and the off-chip bandwidth requirements of its workloads.

11. ACKNOWLEDGMENTS

First and foremost we would like to acknowledge the many authors of the PARSEC benchmark programs which are too numerous to be listed here. The institutions who contributed the most number of programs are Intel and Princeton University. Stanford University allowed us to use their code and data for `facesim`.

We would like to acknowledge the contribution of the following individuals: Justin Rattner, Pradeep Dubey, Tim Mattson, Jim Hurley, Bob Liang, Horst Haussecker, Yemin Zhang and Ron Fedkiw. They convinced skeptics and supported us so that a project the size of PARSEC could succeed.

12. REFERENCES

[1] A. Alameldeen, C. Mauer, M. Xu, P. Harper, M. Martin, and D. Sorin. Evaluating Non-Deterministic Multi-Threaded Commercial Workloads. In *Proceedings of the Computer Architecture Evaluation using Commercial Workloads*, February 2002.

[2] A. Alameldeen and D. Wood. Variability in Architectural Simulations of Multithreaded Workloads. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, February 2003.

[3] P. Banerjee. *Parallel algorithms for VLSI computer-aided design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[4] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, December 1986.

[5] L. Barroso, K. Gharachorloo, and F. Bugnion. Memory System Characterization of Commercial Workloads. In *Proceedings of the 25th International Symposium on Computer Architecture*, pages 3–14, June 1998.

[6] C. Bienia, S. Kumar, and K. Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. In *Proceedings of the 2008 International Symposium on Workload Characterization*, September 2008.

[7] Black, Fischer, and Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81:637–659, 1973.

[8] J. Deutscher and I. Reid. Articulated Body Motion Capture by Stochastic Search. *International Journal of Computer Vision*, 61(2):185–205, February 2005.

[9] P. Dubey. Recognition, Mining and Synthesis Moves Computers to the Era of Tera. *Technology@Intel Magazine*, February 2005.

[10] G. Grahne and J. Zhu. Efficiently Using Prefix-trees in Mining Frequent Itemsets. November 2003.

[11] D. Heath, R. Jarrow, and A. Morton. Bond Pricing and the Term Structure of Interest Rates: A New Methodology for Contingent Claims Valuation. *Econometrica*, 60(1):77–105, January 1992.

[12] L. Hernquist and N. Katz. TreeSPH - A unification of SPH with the hierarchical tree method. *The Astrophysical Journal Supplement Series*, 70:419, 1989.

[13] C. J. Hughes, R. Grzeszczuk, E. Sifakis, D. Kim, S. Kumar, A. P. Selle, J. Chhugani, M. Holliman, and Y.-K. Chen. Physical Simulation for Animation and Visual Effects: Parallelization and Characterization for Chip Multiprocessors. *SIGARCH Computer Architecture News*, 35(2):220–231, 2007.

[14] A. Jaleel, M. Mattina, and B. Jacob. Last-Level Cache (LLC) Performance of Data-Mining Workloads on a CMP - A Case Study of Parallel Bioinformatics Workloads. In *Proceedings of the 12th International Symposium on High Performance Computer Architecture*, February 2006.

[15] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The ALPBench Benchmark Suite for Complex Multimedia Applications. In *Proceedings of the 2005 International Symposium on Workload Characterization*, October 2005.

[16] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Ferret: A Toolkit for Content-Based Similarity Search of Feature-Rich Data. In *Proceedings of the 2006 EuroSys Conference*, pages 317–330, 2006.

[17] K. Martinez and J. Cupitt. VIPS - a highly tuned image processing software architecture. In *Proceedings of the 2005 International Conference on Image Processing*, volume 2, pages 574–577, September 2005.

[18] MediaBench II. `http://euler.slu.edu/~fritts/mediabench/`.

[19] M. Müller, D. Charypar, and M. Gross. Particle-Based Fluid Simulation for Interactive Applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[20] R. Narayanan, B. Özisikyilmaz, J. Zambreno, G. Memik, and A. N. Choudhary. MineBench: A Benchmark Suite for Data Mining Workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization 2006*, pages 182–188, 2006.

[21] L. O'Callaghan, A. Meyerson, R. M. N. Mishra, and S. Guha. High-Performance Clustering of Streams and Large Data Sets. In *Proceedings of the 18th International Conference on Data Engineering*, February 2002.

[22] Pin. `http://rogue.colorado.edu/pin/`.

[23] S. Quinlan and S. D. Venti. A New Approach to Archival Storage. In *Proceedings of the USENIX Conference on File And Storage Technologies*, January 2002.

[24] E. Sifakis, I. Neverov, and R. Fedkiw. Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data. *ACM Transactions on Graphics*, 24(3):417–425, 2005.

[25] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[26] 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*, pages 24–36, June 1995.

[27] G. Xu. A New Parallel N-Body Gravity Solver: TPM. *The Astrophysical Journal Supplement Series*, 98:355, 1995.

[28] T. Y. Yeh, P. Faloutsos, S. Patel, and G. Reinman. ParallAX: An Architecture for Real-Time Physics. In *Proceedings of the 34th International Symposium on Computer Architecture*, June 2007.