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.