SANDIA REPORT

SAND2013-0055
Unlimited Release
Printed January, 2013

An Examination of Content Similarity within the Memory of HPC Applications

Scott Levy, Kurt B. Ferreira, Patrick G. Bridges, Aidan P. Thompson, Christian Trott

Sandia National Laboratories

SAND2013-0055
Unlimited Release
Printed January, 2013

An Examination of Content Similarity within the Memory of HPC Applications

Scott Levy
Patrick G. Bridges
Department of Computer Science
University of New Mexico

Kurt B. Ferreira
Aidan P. Thompson
Christian Trott
Sandia National Laboratories

Abstract

Memory content similarity has been effectively exploited for more than a decade to reduce memory consumption. By consolidating duplicate and similar pages in the address space of an application, we can reduce the amount of memory it consumes without negatively affecting the application's perception of the memory resources available to it. In addition to memory de-duplication, there may be many other ways that we can exploit memory content similarity to improve system characteristics.

In this paper, we examine the memory content similarity of several HPC applications. By characterizing the memory contents of these applications, we hope to provide a basis for efforts to effectively exploit memory content similarity to improve system performance beyond memory deduplication. We show that several applications exhibit significant similarity and consider the source of the similarity.

Introduction

Memory content duplication has been effectively exploited for more than a decade to reduce memory consumption. By consolidating duplicate pages in the address space of an application, we can reduce the amount of memory it consumes without negatively affecting the application's perception of the memory resources available to it. Although this approach has been used to solve problems in a variety of system software domains, it has been most fruitful in the context of virtualization.

Beginning with transparent memory sharing in the Disco Virtual Machine Monitor (VMM) [3], memory deduplication in VMMs has been the subject of significant research efforts as well as commercial development and deployment [17]. Collectively, these endeavors have yielded impressive results. For some applications, memory deduplication across Virtual Machines (VM) has been shown to reduce the total memory footprint by more than 50%.

In addition to consolidating duplicate pages, it is also possible to consolidate similar pages. The Difference Engine was the first to demonstrate that this approach was feasible for memory deduplication [4]. In the Difference Engine, two memory pages are similar if the difference between the two can be captured in a patch file that is smaller than 2 kB. By relaxing the requirement that only duplicate pages be consolidated, the Difference Engine is able to achieve a larger reduction in memory consumption. In some instances, the Difference Engine is able to extract nearly twice as much memory savings as a VMM that consolidates only duplicate pages.

In retrospect, the extent of memory content similarity in virtualization (particularly across VMs running the same guest OS and applications) is perhaps not surprising. Each VM would otherwise maintain its own copies of libraries, instructions and other read-only data. Given these results, memory deduplication and the existence of memory content similarity are well established for virtualization workloads. In contrast, when we consider HPC workloads we tend to believe that memory pages are unique. HPC applications are designed for high performance on large-scale systems; we expect little or no similarity in the memory of these applications. However, *SBLLmalloc* demonstrates that for several HPC workloads significant duplication exists within and across MPI processes [2].

In this paper, we examine the memory content similarity of several HPC applications. By characterizing the memory contents of these applications, we hope to provide a basis for efforts to effectively exploit memory content similarity to improve system performance beyond memory deduplication. We begin, in section 2, by describing our approach for collecting the information that forms the basis of our analysis. In section 3, we present our results. Finally, in section 4, we summarize our observations and conclude.

ASC Sequoia Marquee Performance Codes [9]	AMG	A parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids [5].
	IRS	Implicit Radiation Solver. Solves the radiation transport equation by the flux-limited diffusion approximation using an implicit matrix solution [7].
DOE Production Applications	CTH	A multi-material, large deformation, strong shock wave, solid mechanics code [11]
	LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator. A classical molecular dynamics simulator [14].
Mantevo Mini-Applications [13], [6]	HPCCG	Designed to mimic the finite element generation, assembly and solution for an unstructured grid problem.
	phdMesh	Parallel Heterogeneous Dynamic Mesh. An application designed to mimic the contact search applications in an explicit finite element application.
Miscellaneous Applications	SAMRAI	Structured Adaptive Mesh Refinement Application Infrastructure. Designed to enable the application of structured adaptive mesh refinement to large-scale multi-physics problems [8].
	Sweep3D	Solves a 1-group time-independent discrete ordinates (Sn) 3D cartesian (XYZ) geometry neutron transport problem. [10]

Table 1. A brief summary of HPC applications used

Approach

We examined the memory of the eight HPC applications in Table 1 to ascertain the extent of memory content similarity in HPC applications. We began by placing each page in the address space of an application into one of four categories:

- DUPLICATE PAGES : pages whose contents exactly match one or more other pages and include at least one non-zero byte.
- ZERO PAGES : pages whose contents are entirely zero.
- SIMILAR PAGES : pages that (a) are not duplicate or zero pages; and (b) can be paired with at least one other page in application memory such that the difference between the two can be represented by a `cx_bsdiff` [15] patch that is smaller than 128 bytes.
- UNIQUE PAGES : pages that do not fall into any of the preceding three categories.

For the purposes of the analysis in this paper, we treat zero pages and unique pages identically. In practice, we can treat zero pages as duplicate pages (i.e., if a memory error occurs on a zero page, reconstruction of the damaged page is straightforward). However, to avoid overestimating the impact of our proposed approach, we distinguished zero pages from duplicate pages in our analysis. Zero pages may be an artifact of memory allocation and may not represent memory that is actually being used. For example, to minimize the number of requests for memory from the kernel, `malloc` requests more memory than it actually needs. At least until `malloc` starts recycling pages within the application, its unused pages are zero pages that are not actually resident in memory. And because of the way that we are collecting application data (i.e., reading the entirety of the application's address space), we may be introducing zero pages that would not otherwise exist. Because of these caveats, we believe that categorizing zero pages separately from non-zero duplicates is the clearest and most accurate way to characterize the application behavior we describe here. Finally, we observe that treating zero pages this way is consistent with our intuition; we tend to believe that it is unlikely that an application would actively use large numbers of zero pages.

Evaluation

We generated the data presented in this paper by running each application using MPICH on 8 nodes of a Cray XE6 supercomputer. We used 8 processes on each node for a total of 64 MPI ranks.

Data Collection

We built a library, `libmemstate`, to collect snapshots of the applications' memory and linked it against each of the target applications. The MPI Profiling layer allows us interpose `libmemstate` in all calls by the application to `MPI_Init` and `MPI_Finalize`. By intercepting the call to `MPI_Init`, `libmemstate` is able to snapshot the application's memory after initialization but before the application has started execution. To generate a snapshot of the application's memory, `libmemstate` reads the `/proc/<pid>/maps` file provided by Linux to gather information about the application's address space. Based on the information it gathers, `libmemstate` is able to write a copy of the address space to stable storage.

After the initialization snapshot is complete, `libmemstate` sets a timed signal (`SIGALRM`) that allows it to periodically snapshot memory as the application runs. Unless otherwise noted, we collected all of the data in this paper by configuring `libmemstate` to capture a memory snapshot after every 60 seconds of application execution time.

The process is similar when the application calls `MPI_Finalize`. The MPI Profiling layer interposes a call to `libmemstate`. This allows `libmemstate` to take a finalization memory snapshot and disable its timer.

Each snapshot includes all of the application's heap, stack and anonymous memory. We excluded memory-mapped files because the majority of pages that corresponding to memory-mapped files in the applications that we considered are mapped read-only. The most straightforward way to recover these pages is to re-read their contents from the backing store. As a result, our approach offers little additional protective benefit. However, we can, in practice, use pages backed by stable storage as reference pages for other pages in application memory. But because of this asymmetry, we excluded pages that correspond to memory-mapped files to simplify our analysis.

Data Analysis

After we collected snapshots of the applications' memory, we analyzed them offline. For each snapshot, we walked through the application's virtual address space from low address to high, categorizing each page of memory into one of the four categories described above: (a) duplicate; (b) similar; (c) zero; and (d) unique.

Duplicate Pages

Naively, identifying duplicate pages is a $O(n^2)$ operation. To reduce the cost of identifying duplicate pages, Each collision represents a duplicate page. Although it is conceivable that two or more different pages could yield the same MD5 sum, we assume that the memory contents of the applications we consider are not adversarial. As a result, the probability of such an event is exceedingly small (i.e., $\approx 10^{-14}$) even for very large memory snapshots [18].

Similar Pages

As with identifying duplicate pages, the naive approach to identifying similar pages is an $O(n^2)$ operation. To mitigate this cost, we use an approach inspired by [4]. Instead of computing patches between every pair of pages, we attempt to identify a tractably small set of pages for each candidate page that are likely to be similar to it.

At the outset, we randomly choose four locations, aligned on 128-byte boundaries, in a 4kB page of memory. Once chosen, these four locations are fixed for all of the 4kB pages in the sequence of memory snapshots for a given application. We then create four hash tables that correspond to each of these locations. Each hash table holds a single entry per key. For each candidate page, we compute the MD5 sum of the four 128-byte blocks at the selected locations. The MD5 sums are used to insert the page into the corresponding hash tables. The effect is that, for each block, the associated hash table contains only the most recently examined page for each MD5 sum (i.e., the nearest page with a given MD5 sum that occupies a lower address in the application's virtual address space.) A collision indicates that there is a page that is likely to be similar. This approach identifies up to four pages that may be similar to the current candidate page. In addition to these pages, we also consider the previous candidate page (i.e., the page that occupies the next lowest address in use in the application's virtual address space). In all, this approach identifies as many as five pages that are likely to be similar to the candidate page.

We then compute a patch between the current candidate page and each member of the set of up to five likely similar pages. If any patch is smaller than 128 bytes, we mark the current candidate page as similar. Because `cx_bsdiff` does not generate symmetric patches, observing a single patch that falls below our threshold is sufficient to categorize only a single page as similar. Therefore, we also compute the reciprocal patch of each of the pages in the set of likely similar pages to determine whether any of them should also be marked as similar. As in [4], this is a statistical, heuristic approach. Although there may be more effective ways of identifying similar pages, the fraction of similar pages we identify using this approach is a lower bound on the total number of similar pages in application memory.

Zero Pages

Identifying zero pages is a straightforward process. Although there are many ways that we could identify zero pages, we leverage the process of identifying duplicate pages. Initially, we make no effort to distinguish zero pages from any other page; we insert them into the duplicates hash table as we would any other page. By precomputing the MD5 sum of a 4kB zero page, we can then identify the zero pages as the set of pages that were stored in the duplicates hash table using the zero page MD5 sum as a key.

Unique Pages

Unique pages are those pages that fall outside of the criteria for the preceding three categories. However, we note that this is not a rigid definition; it is highly dependent on our somewhat arbitrary choice of patch size threshold. In particular, increases to the patch size threshold will increase the number of similar pages and decrease the number of unique pages. With a sufficiently large patch size threshold, we could, in principle, transform all of the unique pages into similar pages. In subsequent sections, we examine the tradeoffs involved in changing the patch size threshold.

Repeatability

Non-determinism exists in our methods for collecting and analyzing the data presented here. With respect to data analysis, the source of non-determinism is explicit: as described above, we randomly choose four hash blocks. To begin to understand the variation introduced by this approach, we ran our analysis scripts ten times (randomly choosing four blocks each time) on the memory snapshots collected for LAMMPS. We observed that the number of similar pages varied by less than 0.28% across all of the snapshots (excluding the initialization and finalization snapshots) we collected. This result is likely the product of our very conservative patch threshold. Two pages whose difference can be captured in a 128-byte patch likely differ on a relatively small number of bytes. As a result, we are likely to identify the pages as potentially similar regardless of which set of locations we choose to hash.

With respect to data collection, the timers we use to determine the interval between memory snapshots are not precise. As a result, from run to run we cannot be sure that the snapshots are taken precisely relative to the application's progress. Therefore, it is unlikely that any two sequences of memory snapshots will agree on the exact contents of memory at any given time. Moreover, there may be some variability in the layout of each application's address space that may effect how pages are categorized. To begin to understand the extent of the variability in our data collection mechanism, we collected memory snapshots for ten separate runs of LAMMPS. We then used our analysis scripts to categorize the pages for each sequence of memory snapshots. To control for the variability introduced by our analysis scripts, we fixed the locations of the four hash blocks used in our similarity detection algorithm. We observed that the number of similar pages varied by no more the 2%. We did, however, observe more substantial variability in the number of zero pages. While the number of zero pages varied by several thousand pages in certain cases, these changes were almost entirely offset by changes in the number of unique pages. We believe that this behavior is due to the fact that our memory snapshots are not taken at precisely the same point in the application's execution each time.

While these results are not exhaustive, they do suggest that the data we collected is representative of the memory content behavior of the applications that we considered.

Results

We devote this section to presenting and analyzing the data that we collected from each application. For each application, we examine the set of memory snapshots that we collected to illuminate the nature of memory content similarity within the application's memory.

Overview

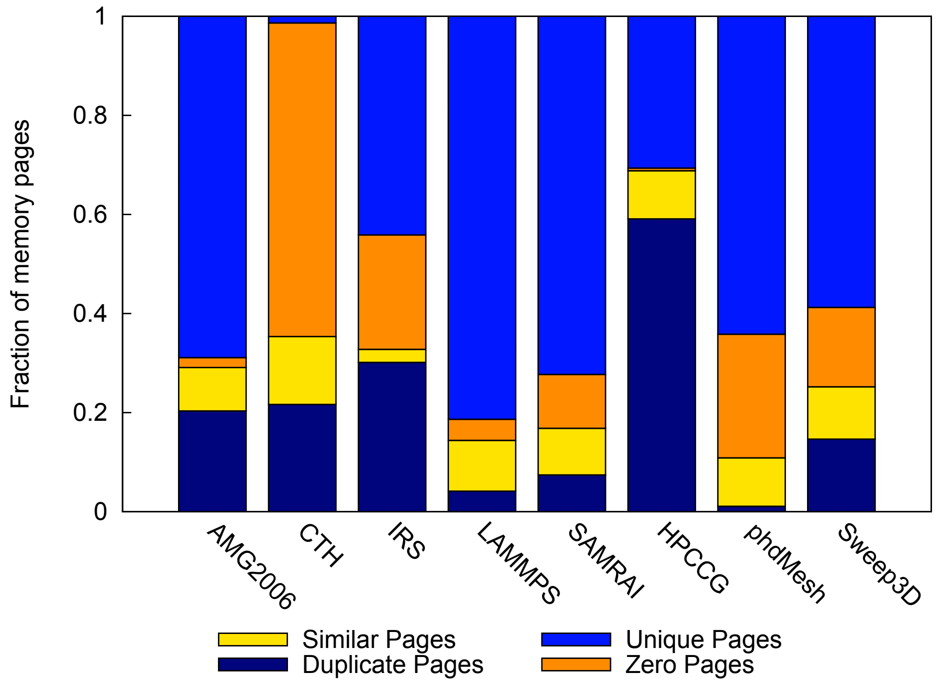


Figure 1. Page categorization within Rank 0 for each application. Each bar represents the page categorization for the memory snapshot that contained the smallest fraction of similar and duplicate pages.

We begin by considering the categorization of the memory pages in the virtual address space of each application. Figure 1 illustrates memory content similarity, expressed as a fraction of application memory, for each application. Each bar represents the memory snapshot (excluding the initialization and finalization snapshots) that contains the smallest fraction of similar and duplicate pages that we observed for each application. Complete memory categorization for the entire lifetime of each application can be found in Figures 8-15.

Overall, these figures demonstrate that significant similarity exists in half of the applications we considered. Between 25 and 35% of the pages in the memory of rank 0 for AMG, IRS, CTH and Sweep3D are either duplicate or similar. LAMMPS, SAMRAI and phdMesh exhibit more modest similarity with as little as 13.4%, 16.8% and 10.8% of application memory devoted to similar and duplicate pages, respectively. However, even given these modest results, there may still be opportunities to exploit similarity in these applications, particularly if we consider patch sizes larger than 128 bytes.

HPCCG is a clearly an outlier. More than two thirds of the pages in its virtual address space are similar or duplicate. This is substantially higher than the application with the next highest fraction. Given these results and the fact that HPCCG is a mini-application, we are inclined to discount this result.

Temporally, we observe that the trends in content similarity exhibited by these applications fall into three broad categories: (a) Stable; (b) Noisy; and (c) Dynamic. Examples of the behavior that defines each of these categories is shown in Figure 2. LAMMPS, IRS, HPCCG, phdMesh and Sweep3D constitute the temporally stable category. Excluding the initialization and finalization snapshots, the virtual address space of each these applications includes a stable fraction of duplicate and similar pages. The temporally erratic class of applications consists of CTH and SAMRAI. These two applications show significant fluctuations in the number of duplicate and similar pages in their virtual address spaces. The number of duplicate pages in SAMRAI spikes twice during its run. At one point, the number of duplicate pages nearly triples. CTH exhibits similar, but less pronounced, behavior. Early in its execution, the number of duplicate pages drops by more than 10%. Although the snapshots of these applications captured only a handful of deviations, it suggests that the memory contents of these applications may be more dynamic and unpredictable than the other applications we considered. Finally, the duplication and similarity in the memory AMG2006 exhibits significantly different behavior than any of the other applications. For approximately the first half of its execution, the fraction of duplicate and similar pages in the memory of AMG2006 steadily decreases before stabilizing for the remainder of its run.[1] To account for this behavior, we define a third category: temporally dynamic applications.

NUMA

We ran all of our tests on a Cray XE6 system. Because each of the XE6 compute nodes uses a NUMA architecture, we may be able to increase similarity by considering memory across processes. Although it may also be possible to exploit sharing across NUMA domains, the costs of maintaining similarity information is likely to be much higher. As a result, there are likely many more opportunities to effectively exploit memory content similarity within a NUMA domain. Therefore, throughout this document, we only consider similarity across processes within a single NUMA domain.

Each compute node of the XE6 contains two 8-core AMD Opteron Magny-Cours processors. Each Magny-Cours processor is divided into two NUMA domains. And each NUMA

[1]We also observe that AMG is the only application that allocates significant quantities of memory after initialization

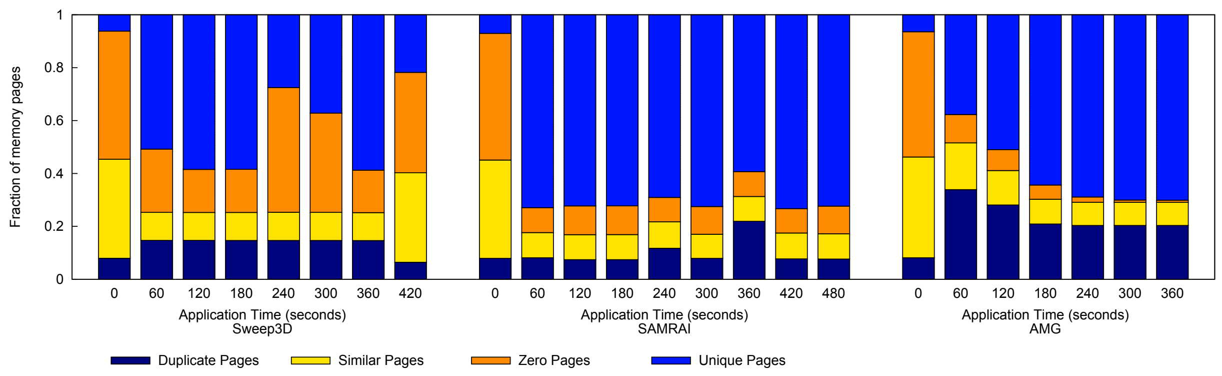


Figure 2. Examples of notable temporal trends in memory content similarity: Stable (IRS), Erratic (SAMRAI) and Dynamic(AMG)

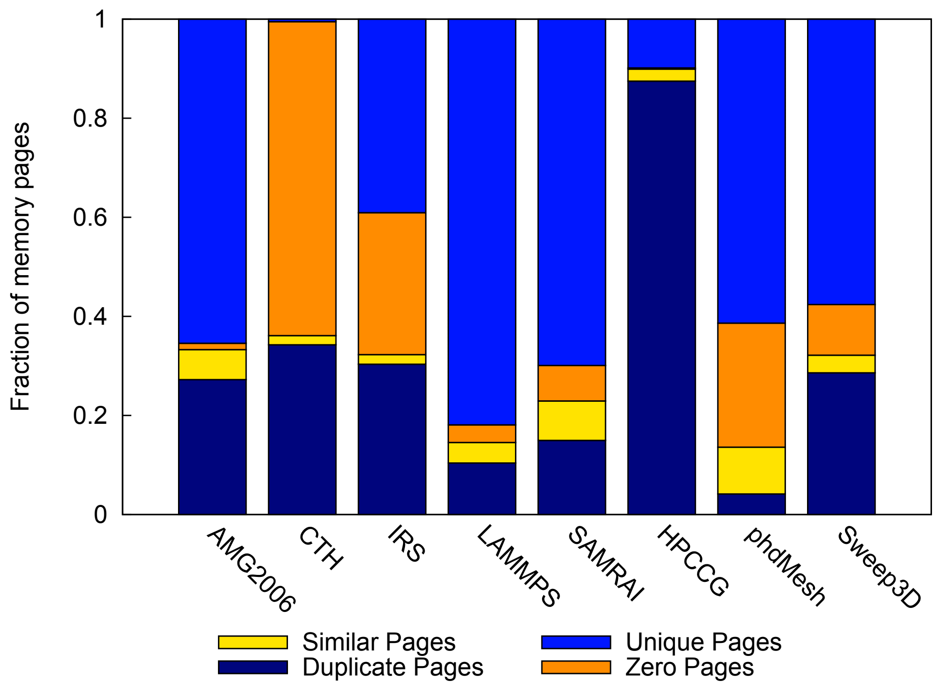


Figure 3. Page categorization within a NUMA domain as a function of application time.

domain is comprised of four cores [16]. We use the default MPICH layout method which results in SMP-style placement of MPI ranks. Based on this architecture, we were able to group our memory snapshots by rank to effectively examine content similarity within a

Application	Rank 0 Duplicate	Rank 1 Duplicate	Rank 2 Duplicate	Rank 3 Duplicate	Total Duplicate	NUMA Domain Duplicate	% Increase
AMG2006	43820	47911	49173	44215	185119	234162	26.5 %
CTH	11573	9470	10153	9311	40507	63583	57.0 %
IRS	13769	13863	13829	13774	55235	59320	7.4 %
LAMMPS	3534	3537	3566	3662	14299	35541	148.6 %
SAMRAI	1174	518	1920	391	4003	7437	85.8 %
HPCCG	131143	134229	144576	147495	557443	778302	39.6 %
phdMesh	828	2842	2198	2137	8005	13921	73.9 %
Sweep3D	844	844	844	844	3376	6183	83.1 %

Table 2. Number of duplicate pages in the middle snapshot for each application

NUMA domain for each application. The results are shown in Figure 3. As with the single rank results, these figures also show the total memory usage of each application as a function of time.

The effect of considering similarity across all of the processors in a NUMA domain varies considerably among the applications that we consdiered. Tables 2, 3 and 4 break down the effect in more detail. Table 2 shows the number of duplicate pages for each of the ranks in a NUMA domain when they are considered individually and collectively. Similarly, Table 3 shows the number of similar pages for each of the ranks in a NUMA domain when they are considered individually and collectively. Table 4 describes the overall effect of expanding the scope of our similarity analysis to include an entire NUMA domain.

Three categories emerge from this data. First, Sweep3D, AMG, SAMRAI and HPCCG see significant benefit from jointly considering all of the ranks in a NUMA domain. Second, we observe more modest gains in phdMesh. Third, in the address space of LAMMPS, CTH and IRS, the number of duplicate and similar pages changes little when the ranks in a NUMA domain are considered together.

We broadly observe that expanding the scope of the application memory that we consider to include a NUMA domain tends to significantly increase the number of duplicate pages at the expense of similar pages. This effect is especially pronounced in LAMMPS and CTH; we observe a sizeable increase in duplicate pages (148.6% and 57.0%, respectively) when the NUMA domain is considered. However, these gains are almost entirely offset by a decrease in similar pages. As a result, for these two applications, if we can effectively exploit similar pages (and bear the metadata costs) there may be little benefit to considering the ranks of a NUMA domain collectively.

Application	Rank 0 Similar	Rank 1 Similar	Rank 2 Similar	Rank 3 Similar	Total Similar	NUMA Domain Similar	% Increase
AMG2006	19614	13272	14087	16985	63958	53185	-16.8 %
CTH	6427	6663	5905	6462	25457	3737	-85.3 %
IRS	1195	1199	1285	1290	4969	2904	-41.6 %
LAMMPS	8744	8787	8855	8922	35308	14344	-59.4 %
SAMRAI	1002	1523	1062	1743	5330	3202	-39.9 %
HPCCG	21553	26747	26754	26753	101807	21569	-78.8 %
phdMesh	7483	9501	9078	9161	35223	31976	-9.2 %
Sweep3D	609	612	611	608	2440	812	-66.7 %

Table 3. Number of similar pages in the middle snapshot for each application

Address Space Behavior

Given the observations that we have made regarding memory content similarity to this point, it is instructive to consider where this similarity is coming from. To begin to answer this question, Figure 4 contains a series of heat maps of the virtual address space of IRS. Figures 16-24 contain a similar set of heat maps for the other applications that we considered. In each heat map, we have divided the application's address space into 256 buckets. Each horizontal line in the heat maps represents one bucket of the application's address space over the lifetime of the application. In Figure 4(a), the heat map represents the fraction of the application's address space for rank 0 that is comprised of duplicate and similar pages. Likewise, Figure 4(b) represents the fraction of duplicate and similar pages in the address space of rank 0 when the three other ranks that jointly constitute a NUMA domain are considered collectively.

In these two subfigures, the areas of dark orange represent buckets that contain a large fraction of duplicate and similar pages. The areas of yellow represent buckets that contain few duplicate or similar pages. As the color changes from yellow to orange, it indicates an increase in the fraction of duplicate and similar pages in a particular bucket. The blue portions of the map indicate virtual addresses that were not part of the application's address space when a given snapshot was taken. The most compelling feature of Figures 4(a) and 4(b) is the presence of distinct horizontal bands of color. In general, we observe very little variation in color from left to right. This trend suggests that pages that are initially classified as similar or duplicate remain similar or duplicate throughout the lifetime of the application. If the classification of individual pages changes infrequently, it may also mean that the pages

Application	Rank 0-3 Total	NUMA Domain	% Increase
AMG2006	249077	287347	15.4 %
CTH	65964	67320	2.1 %
IRS	60204	62224	3.4 %
LAMMPS	49607	49885	0.6 %
SAMRAI	9333	10639	14.0 %
HPCCG	659250	799871	21.3 %
phdMesh	43228	45897	6.2 %
Sweep3D	5816	6995	20.3 %

Table 4. Number of duplicate and similar pages in the middle snapshot for each application

also change infrequently.[2]

Infrequent modification of similar and duplicate pages is important because it reduces the overhead of maintaining similarity information. To effectively exploit memory content similarity, we need to maintain metadata about which pages are duplicates of or similar to each other. Each time a similar or duplicate page is modified, the similarity metadata needs to be updated to account for the effect of the change. If the pages change infrequently, then metadata maintenance will be less disruptive of the application.

To help answer the question of how often similar or duplicate pages change, Figure 4(c) represents how the contents of application memory change over time for IRS. The address space in this subfigure, as in the two preceding subfigures, has been divided into 256 buckets. Here, as the color changes from yellow to orange, it represents an increase in the fraction of pages whose contents have changed since the previous snapshot was taken. At a high level, we observe that the dark orange bands (indicating large fractions of similar and duplicate pages) in Figures 4(a) and 4(b) correspond to regions of yellow (indicating small fractions of changed pages) in Figure 4(c). This is a promising trend; it suggests that similar and duplicate pages change relatively infrequently.

Heat maps and change maps for all of the other applications that we considered are available in Figures 16-24 of the Appendix. With the exception of SAMRAI and CTH, we see very similar trends in the heat maps of the other applications; bands of orange in the similarity heat maps correspond to regions that are relatively static. Based on this analysis, CTH and SAMRAI are the most problematic of the applications that we considered. In Figures 17 and 20, we observe that the color of a single bucket varies, sometimes dramati-

[2]We note that it is possible for a similar or duplicate page to be recategorized even if its contents of the page itself are not altered. This can occur, for example, if the page to which a given page is similar is modified.

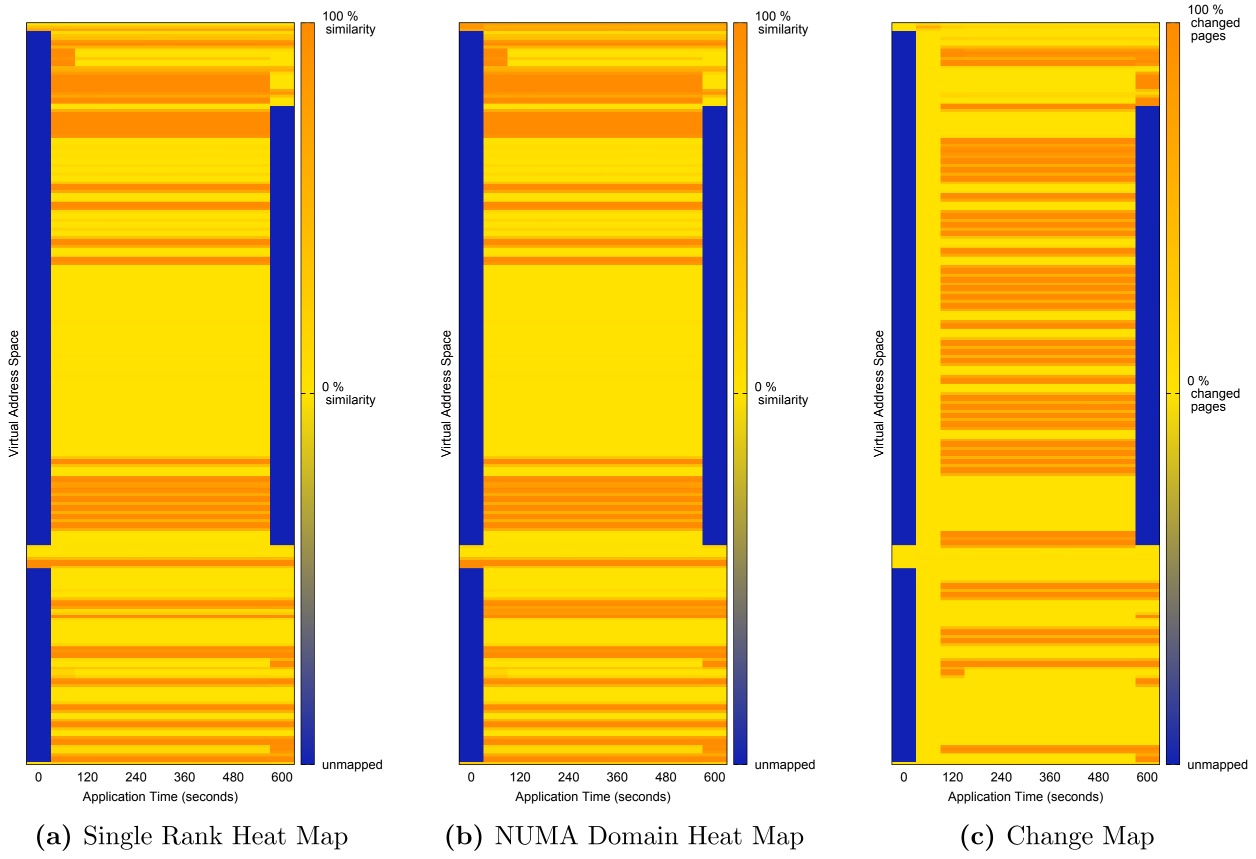
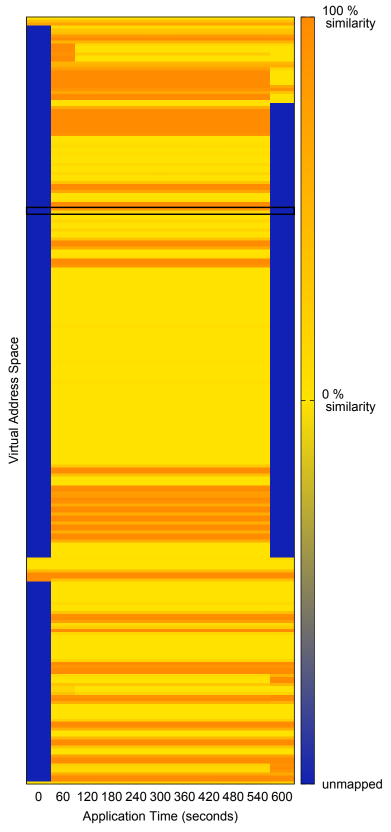


Figure 4. Address space behavior of IRS as a function of application time.

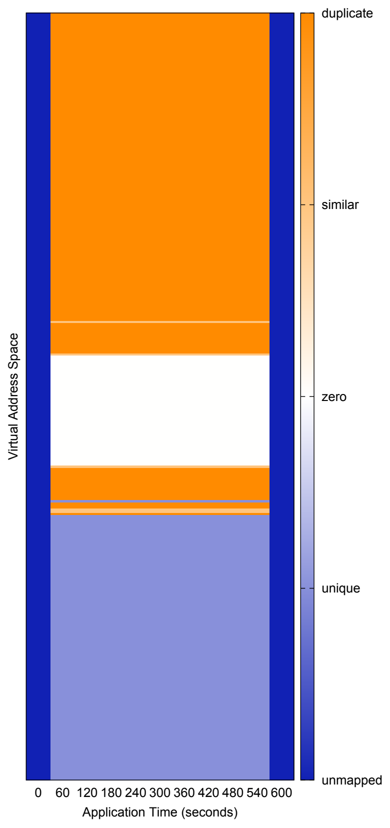
cally, over the application's lifetime. This suggests that these applications will require more frequent maintenance of their similarity metadata.

In an effort to more fully understand the modification behavior of similar and duplicate pages, Figure 5 presents a more detailed examination of a portion of the address space of IRS. Figure 5(a) is the same as Figure 4(a) with one small change. About a quarter of the way down the virtual address space is a small black rectangle. This rectangle represents the excerpt of the virtual address space that we consider in detail in the remaining subfigures. In Figure 5(b), each horizontal line represents a single page from the excerpt. Each page is colored based on its categorization. Likewise, in Figure 5(c) each horizontal line represents the modification history of a single page.

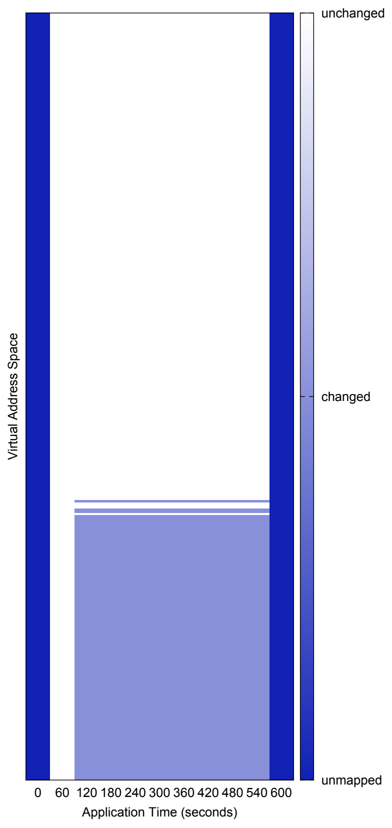
A distinct trend emerges from the detailed heat maps in Figures 5(b) and 5(c). As indicated in the coarse-grained heat maps considered above, we observe that duplicate and similar pages appear to change infrequently over the lifetime of the application. To explore



(a) Full Heat Map. The small black rectangle near the top of the figure highlights the portion of the address space considered in (b) and (c)



(b) Detailed Heat Map



(c) Detailed Change Map

Figure 5. Address space behavior of IRS as a function of application time.

modification behavior in more detail, Figure 5 shows similarity and modification trends for individual pages within a portion of the address space of rank 0 of IRS. Figures 5(b) and 5(c) are almost mirror images of each other. To see this, consider the bottom of the the two figures; each are dominated by a light blue region. The light blue region in Figure 5(b) represents unique pages. The corresponding light blue region in Figure 5(b) represents pages that have changed. Similarly, the upper portions of the address space are dominated by orange regions in Figure 5(b) that represent duplicate and similar pages while the corresponding white regions in Figure 5(c) respresent pages whose contents have not changed. In this detailed look at the address space of IRS, we have a clear example of similar and duplicate pages that remain unchanged while unique pages are modified frequently.

Finally, we consider the modification behavior of of the memory used by these eight applications quantitatively. Table 5 illustrates how frequently similar and duplicate pages

Application	Changed 1+ Times	Changed 1 Time	Changed 2 Times	Changed 3 Times	Changed 4+ Times
AMG2006	20.2 %	12.0 %	3.8 %	3.0 %	1.5 %
CTH	39.1 %	6.5 %	3.5 %	14.1 %	15.0 %
IRS	31.7 %	17.2 %	0.1 %	0.0 %	14.4 %
LAMMPS	11.0 %	0.2 %	0.1 %	0.0 %	10.7 %
SAMRAI	82.2 %	16.0 %	6.9 %	35.9 %	23.4 %
HPCCG	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %
phdMesh	6.9 %	1.7 %	0.4 %	0.3 %	4.4 %
Sweep3D	2.5 %	1.4 %	0.6 %	0.0 %	0.5 %

Table 5. Modification behavior of the pages in the memory of Rank 0 that are ever categorized as similar or duplicate.

change. We begin by noting that for the purposes of this discussion, similar or duplicate pages are those pages that are *ever* categorized as similar or duplicate. In the second column of the table, we see the fraction of similar or duplicate pages that are modified during the lifetime of the application. The subsequent columns break this fraction down based on the number of times that a page changes.

The most striking feature of the data in this table is that for the majority of the applications that we considered, more than two-thirds of the duplicate and similar pages are never modified. Moreover, for LAMMPS, HPCCG, phdMesh and Sweep3D, more 85% of the similar and duplicate pages are static throughout the lifetime of the application. However, SAMRAI, CTH and, to a lesser extent, IRS, are more problematic. More than 80% of the similar and duplicate pages in SAMRAI are modified. Further, nearly two-thirds of all of the similar and duplicate pages were changed three or more times. CTH sees substantially fewer modifications, but nonetheless nearly a third of the similar and duplicate pages were modified three or more times. This suggests that the overhead of maintaining similarity metadata for these applications will be significant.

We observe that, for many of the applications that we considered, similar or duplicate pages rarely change. Although we lack the application-specific knowledge to know definitively the cause of this behavior, the data presented here suggests that the majority of similar or duplicate pages are comprised of read-only or read-mostly data.

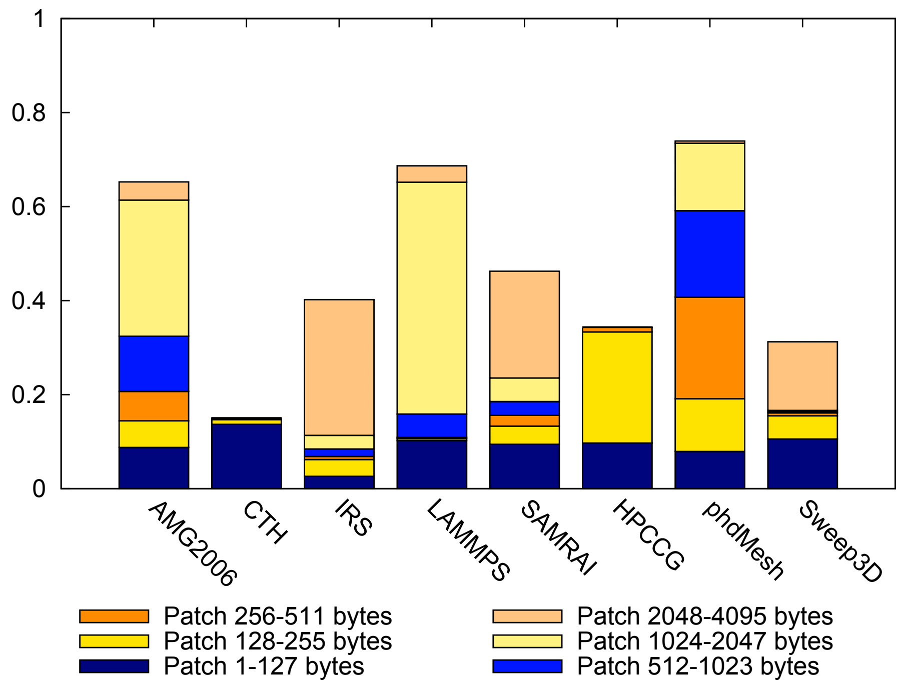


Figure 6. Patch size distribution for rank 0 based on the memory snapshot of each application that contains the smallest fraction of similar and duplicate pages. Pages whose minimum patch size is less that 128 bytes are the pages that we have categorized as Similar Pages throughout this document.

Patch Behavior

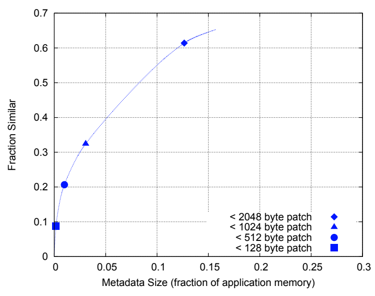
Throughout this document, we have considered the effect of defining similar pages based on a patch size threshold of 128 bytes. This threshold represents a balance struck between maximizing similarity and minimizing the size of metadata that must be maintained. To characterize the benefits of increasing similarity by increasing the patch size threshold, Figure 6 illustrates the benefit of increasing the patch size. As we allow for larger and larger patches the fraction of pages that are classified as similar increases. We note that in some cases, `cx_bsdiff` generates a patch that is larger than a page. For the purposes of this discussion, we consider such pages to be unique.[3]

In Figure 6, three categories emerge: (a) applications for which the patches are almost exclusively small (CTH and HPCCG); (b) applications that are dominated by large patches (IRS, LAMMPS, SAMRAI and Sweep3D); and (c) applications for which the patches are more or less evenly distributed (AMG2006 and phdMesh). For CTH and HPCCG, not

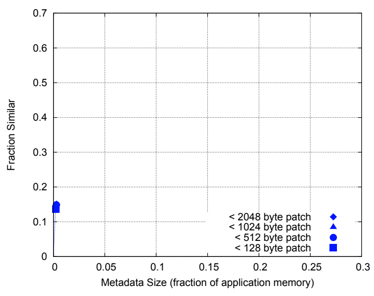
[3]Although we are treating these pages as unique in our analysis here, in practice it is straightforward to capture the difference between any two pages in a single page.

only are the patches are relatively small but the number of potentially similar pages is also relatively small. As a result, we expect the cost (in terms of metadata) of increasing the patch size to extract additional similarity to be relatively small. On the other hand, for the applications that are dominated by large patches, we expect the cost of a increasing the patch size to grow rapidly. Finally, the relatively even distribution of patch sizes in AMG2006 and phdMesh suggests that gains in similarity may be possible for relatively small increases in total metadata.

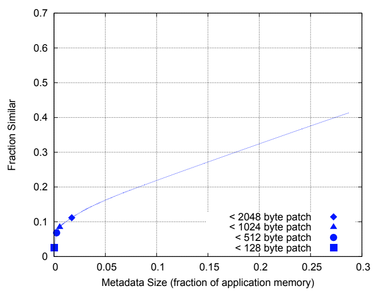
Metadata Costs



(a) AMG2006



(b) CTH



(c) IRS

Figure 7. Representative plots of the three categories of metadata cost trends.

As shown in the preceding section, increasing the patch size threshold enables more similarity to be extracted from the application's memory. The primary cost of increasing the patch size threshold is that the quantity of metadata that must be maintained also increases. As we examine the fraction of similar pages in application memory as a function of metadata size three categories emerge. In Figure 7, we provide plots for applications that representative of these three categories. For the snapshot containing the smallest fraction of similar and duplicate pages, these plots characterize the nature of the tradeoff between the increase in similar (but not duplicate) pages and the required metadata. To illustrate how these tradeoffs relate to specific patch size thresholds, we have plotted the results for four different patch size thresholds ranging from 128 bytes to 2048 bytes. The slope of these curves indicate how expensive it is to extract additional similarity.

A steep curve, as in Figure 7(a), indicates that extracting additional similarity is relatively inexpensive. AMG, LAMMPS and phdMesh fall into this category. 25(b), 28(b), 31(b) For these applications, increasing the patch size threshold results in significant increases in the number of similar pages while imposing a relatively modest cost. For example, if we allowed the metadata for AMG to occupy up to 5% of application memory, the total number of similar and duplicate pages in AMG would approach 60%. For a similar cost, the number

Application	Similar and Duplicate Pages			
	ORIGINAL		WITH METADATA INCREASE	
AMG2006	61037	(29.1 %)	125290	(59.8 %)
CTH	16134	(35.3 %)	16746	(36.7 %)
IRS	14964	(32.8 %)	21146	(46.3 %)
LAMMPS	12275	(14.4 %)	27418	(32.1 %)
SAMRAI	1640	(16.8 %)	3306	(34.0 %)
HPCCG	152694	(68.8 %)	207429	(93.5 %)
phdMesh	8349	(10.9 %)	42651	(55.8 %)
Sweep3D	1457	(25.3 %)	2067	(35.9 %)

Table 6. Effect of Increasing Metadata Size. This table quantifies the increase in similar and duplicate pages when the metadata is allow to occupy 5% of application memory. This data is based on the memory snapshot for each application that contains the smallest fraction of similar and duplicate pages.

of similar and duplicate pages in LAMMPS would rise to nearly 50%. Complete results for allowing each application's metadata to grow to 5% of application memory are shown in Table 6.

However, for small increases in metadata, the cost curve for SAMRAI is relatively steep and so we obtain inexpensive benefits. As a result, expanding to 5% of system memory yields low-cost benefits. Beyond 5% of application memory, the curve flattens out and subsequent gains are increasingly expensive.

For CTH, HPCCG, there is little cost and very modest benefit to increasing the patch size threshold. Even for a very large patch threshold, the metadata will never As a result, for these applications we would likley want to set a very high patch threshold. Finally, we observe that our choice of 128 bytes as the patch size threshold has clearly struck the balance in favor of minimizing metadata. For each application, the metadata necessary to manage patches of this size is below 1% of application memory.

Related Work

Memory content similarity has been explored for more than a decade. As a result, a significant body of relevant research has emerged. However, to our knowledge, we are the first to consider the existence of both similar and duplicate pages in the memory of scientific applications. Moreover, in the context of similar pages, ours is the first examinzation of the costs and benefits of increasing the patch size threshold. Additionally, ours is the first to consider the source of similar and duplicate pages in the memory of scientific applications.

To date, memory content similarity has been most thoroughly explored in the context of data de-duplication. Although de-duplication has been examined in several contexts, the preponderance of the relevant research has been in virtualization. In [3], the authors introduced the concept of transparent memory sharing in VMMs. By intercepting disk requests that DMA data into memory, the Disco VMM could consolidate read-only pages (e.g., text segments of applications, read-only pages in the buffer cache[4]) containing data from the disk across virtual machines. In some cases, this approach allowed the Disco VMM to signficantly reduce memory consumption. For example, transparent memory sharing allowed the VMM to reduce the total memory consumed by 8 VMs, each running the same guest OS and workload, by more than half.

More recently, [17] described the broader approach to memory de-duplication that is used in the VMware ESX server. Instead of intercepting disk requests, the authors propose identifying all pages in a virtual machine by their contents. When any two pages are found to have the same contents, the pages are consolidated using copy-on-write (COW). Applying this approach to systems running as many as 10 identical VMs running the SPEC95 benchmark on Linux, the VMware ESX server is able to reduce memory consumption by nearly 60%.

The data we present here differs from these early de-duplication studies in a couple of key respects. First, the applications considered in these studies have been those that are most commonly used in practice with virtualization (e.g., web servers, databases and development applications). Moreover, these studies examine applications running independently on individual virtualized workstations. In contrast, we considered a suite of scientific applications running on a single HPC system. Second, these virtualization studies explore sharing within and across entire VMs. They allow for sharing between all of the applications running on a VM as well as with the kernel. Our data was collected in the absence of virtualization and represents content similarity that exists only within application memory. Third, in addition to duplicate pages, we also identified pages that were similar but not identical.

The authors of [18] advocate broadening the scope of sharing in virutalization to consider intranode sharing. To evaluate the feasibility of this approach, they consider the prevalence of intranode sharing between nodes running several HPC applications. For some workloads (notably HPCCG), they observe that significant inter- and intra-node sharing opportunities exist. Based on these promising results, they propose a Content-Sharing Detection System for exploiting intranode sharing in virtualized environments. Similarly, *SBLLmalloc* has been used to demonstrate that memory consumption can be significantly reduced by consolidating duplicate pages in the application memory of several HPC applications [2]. In several cases, this approach yields memory savings in excess of 50%. Although we consider a different set

[4]Although the function of the buffer cache has since been folded into the page cache, this term reflects time period in which this paper was written

of applications than [18] and [2], the most striking difference between our work and theirs is that the data presented here in our paper presents a broader view of memory content by examining both duplication and similarity in application memory.

Most memory de-duplication research has considered consolidating only duplicate pages. However, the Difference Engine [4] introduced the idea that similar pages could also be consolidated. In this context, two pages are similar if the difference between them can be represented by an `xdelta` patch file that is smaller than 2kB. By relaxing the requirement that only duplicate pages be consolidated, the authors show that, under some e-commerce workloads, the Difference Engine can extract significantly more memory savings than the VMware ESX server. Moreover, they show that the Difference Engine can reduce memory consumption by more than 50% even for VMMs hosting a single VM. The data presented in [4] was collected by modifying a Xen VMM and using it to host virtualized workstations running workloads consisting of a mix of web and database server and compilation benchmarks. In contrast, we collected our data using by running several scientific applications natively (i.e., without virtualization) on a single HPC system. Our results indicate that a significant fraction of similar pages exist even when we exclude kernel memory and the memory of ancilliary applications and only consider the memory of a single application. Moreover, we have shown that similar pages exist even in scientific applications that have been carefully designed for high performance on tightly-coupled systems.

In addition to virtualization, content duplication has been effectively exploited in other domains. In context of data storage, reducing storage requirements in primary and archival data storage applications by eliminating duplicate data blocks has been widely studied [12], [20], [19]. Kernel Shared Memory (KSM) allows duplicate memory to be consolidated in Linux with or without virtualization [1].

Conclusion

In this paper, we have presented the results of a detailed study of the application memory of eight HPC applications. To the extent possible, we have considered the source of similar and duplicate pages. We have also analyzed the effect of considering all of the application memory in a NUMA domain. Additionally, we have presented data on the costs and benefits of using different patch size thresholds. We discovered several interesting characteristics of these applications. Based on the results and analysis presented above, we draw four conclusions.

- Several HPC applications, including AMG, CTH, IRS and Sweep3D exhibit significant memory similarity.
- Allowing similarity metadata to occupy a modest fraction (e.g., 5-10%) of application memory would increase memory content similarity for several applications.

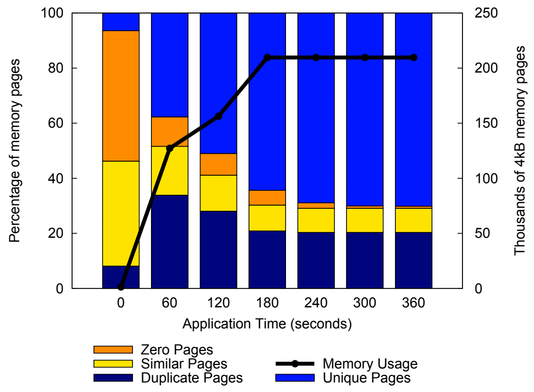
- Counterintuitively, considering all of the application memory in a NUMA domain does not always yield more similar and duplicate pages.
- Similar and duplicate pages appear to be either read-only or read-mostly.

We believe that these results demonstrate that, for a meaningful fraction of HPC applications, significant memory similarity exists and that it may be possible to effectively exploit it.

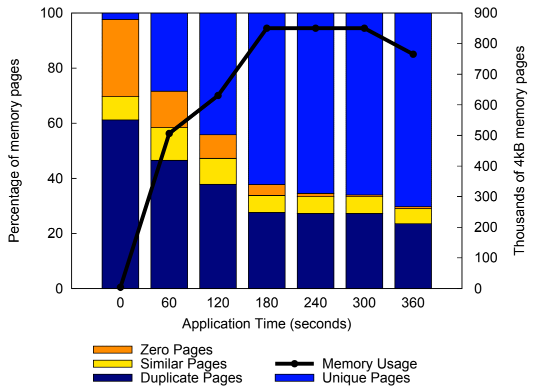
Appendix

Temporal Page Characterization

This section contains histograms showing how the categorization of application memory pages changes over time. To put the results in the proper context, these figures also show total memory used as a function of time.

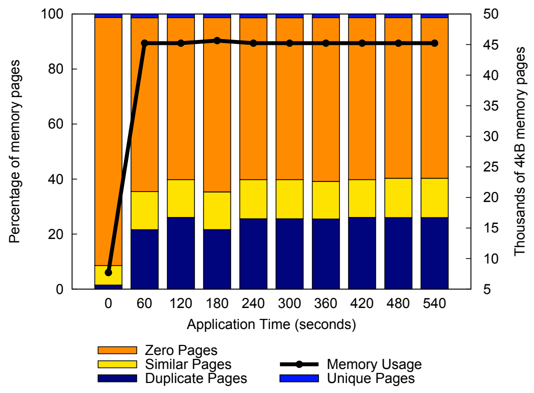


(a) Rank 0

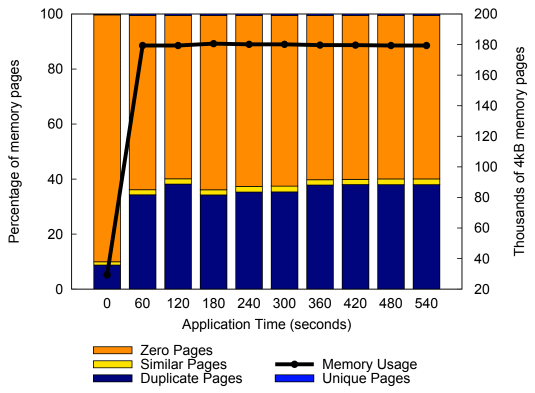


(b) NUMA

Figure 8. Page categorization in AMG2006 as a function of application time.

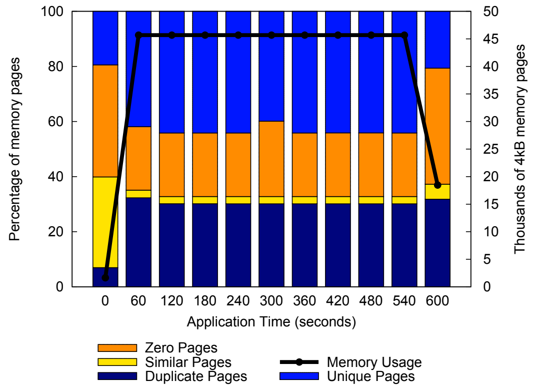


(a) Rank 0

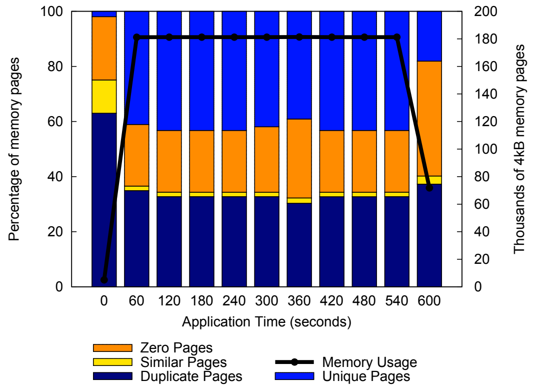


(b) NUMA

Figure 9. Page categorization in CTH as a function of application time.

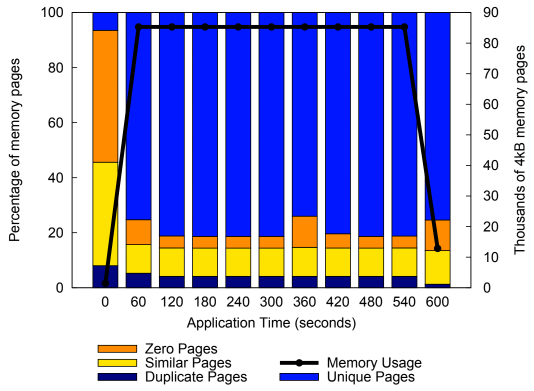


(a) Rank 0

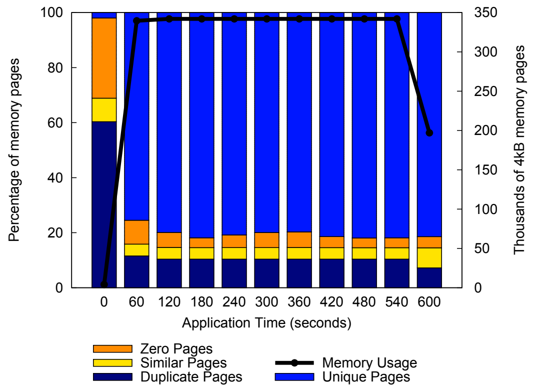


(b) NUMA

Figure 10. Page categorization in IRS as a function of application time.

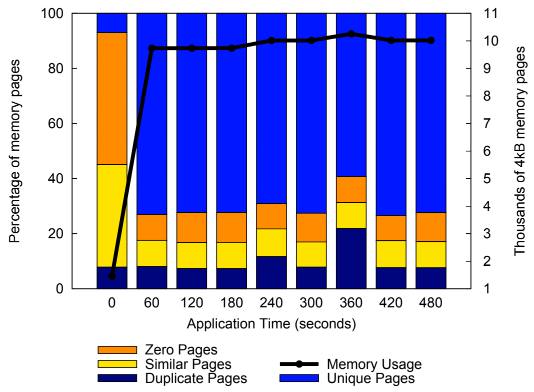


(a) Rank 0

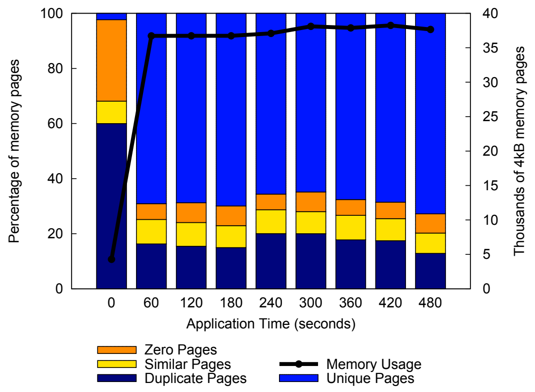


(b) NUMA

Figure 11. Page categorization in LAMMPS as a function of application time.

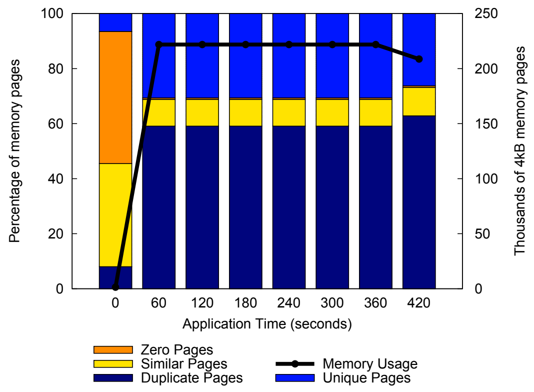


(a) Rank 0

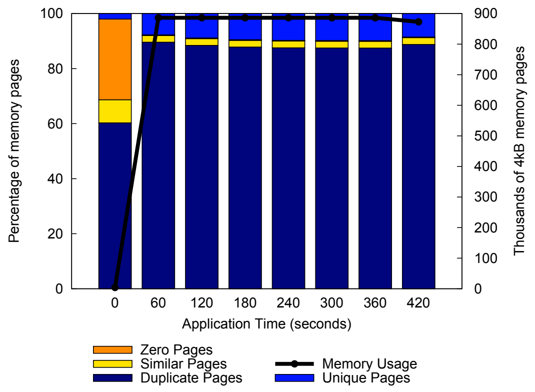


(b) NUMA

Figure 12. Page categorization in SAMRAI as a function of application time.

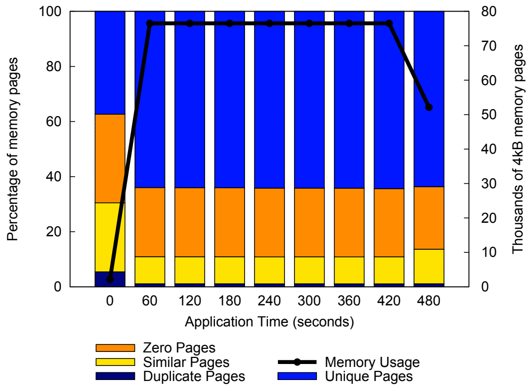


(a) Rank 0

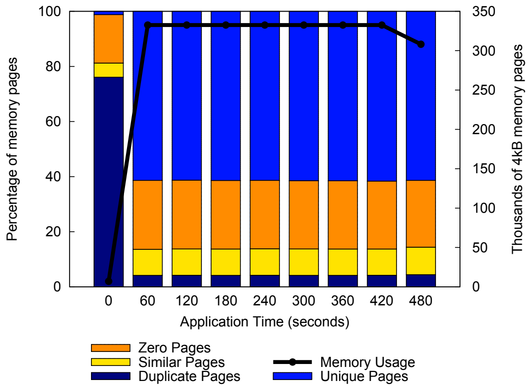


(b) NUMA

Figure 13. Page categorization in HPCCG as a function of application time.

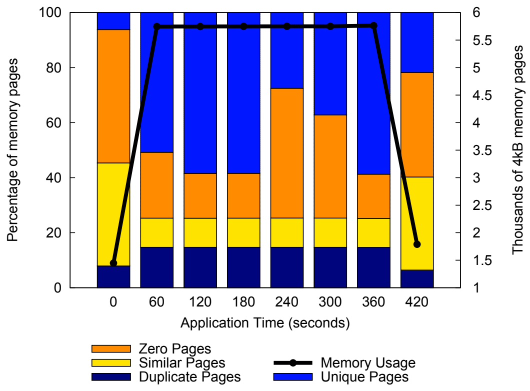


(a) Rank 0

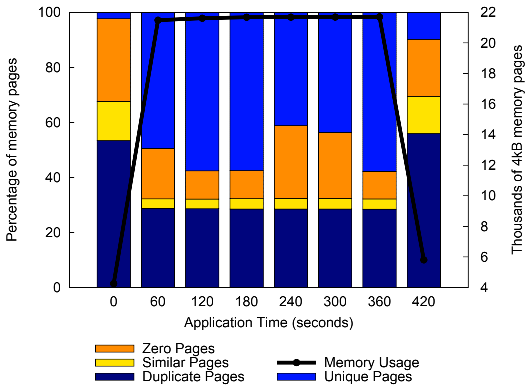


(b) NUMA

Figure 14. Page categorization in phdMesh as a function of application time.



(a) Rank 0

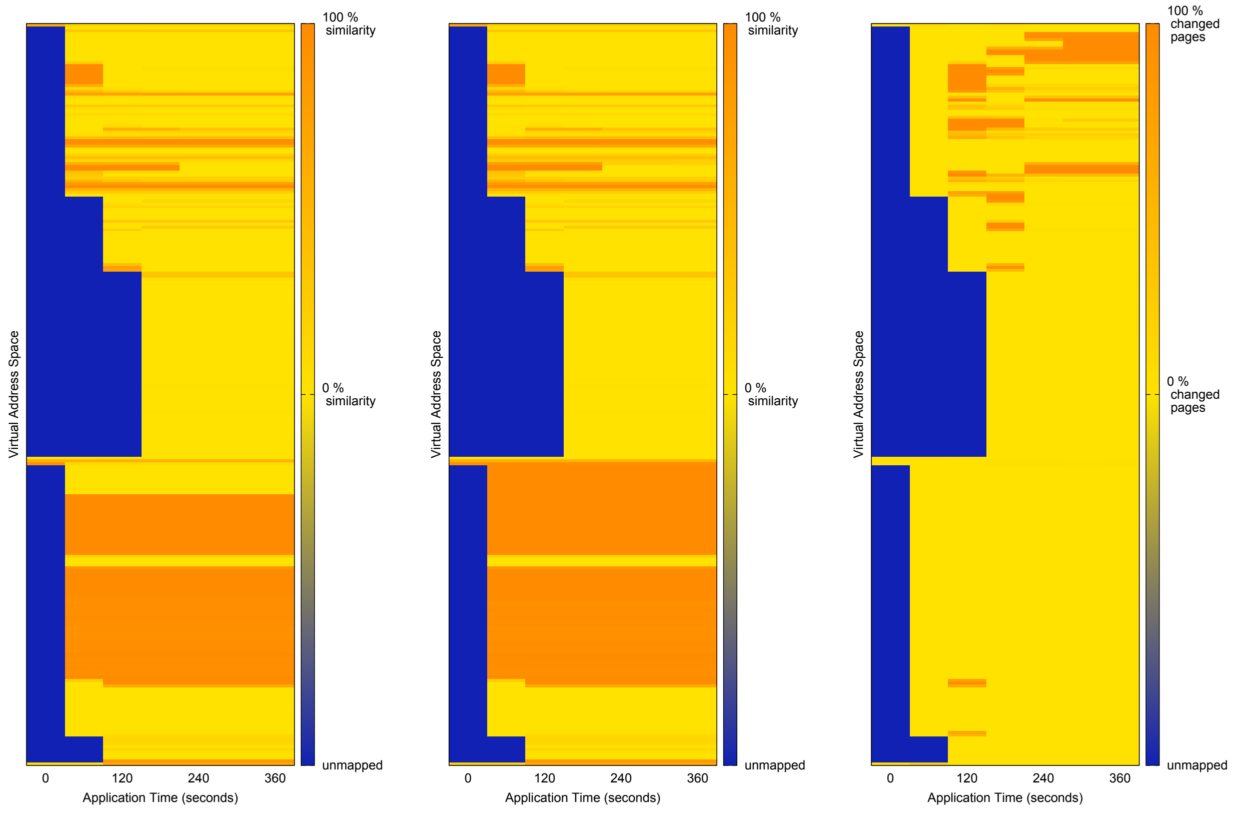


(b) NUMA

Figure 15. Page categorization in Sweep3D as a function of application time.

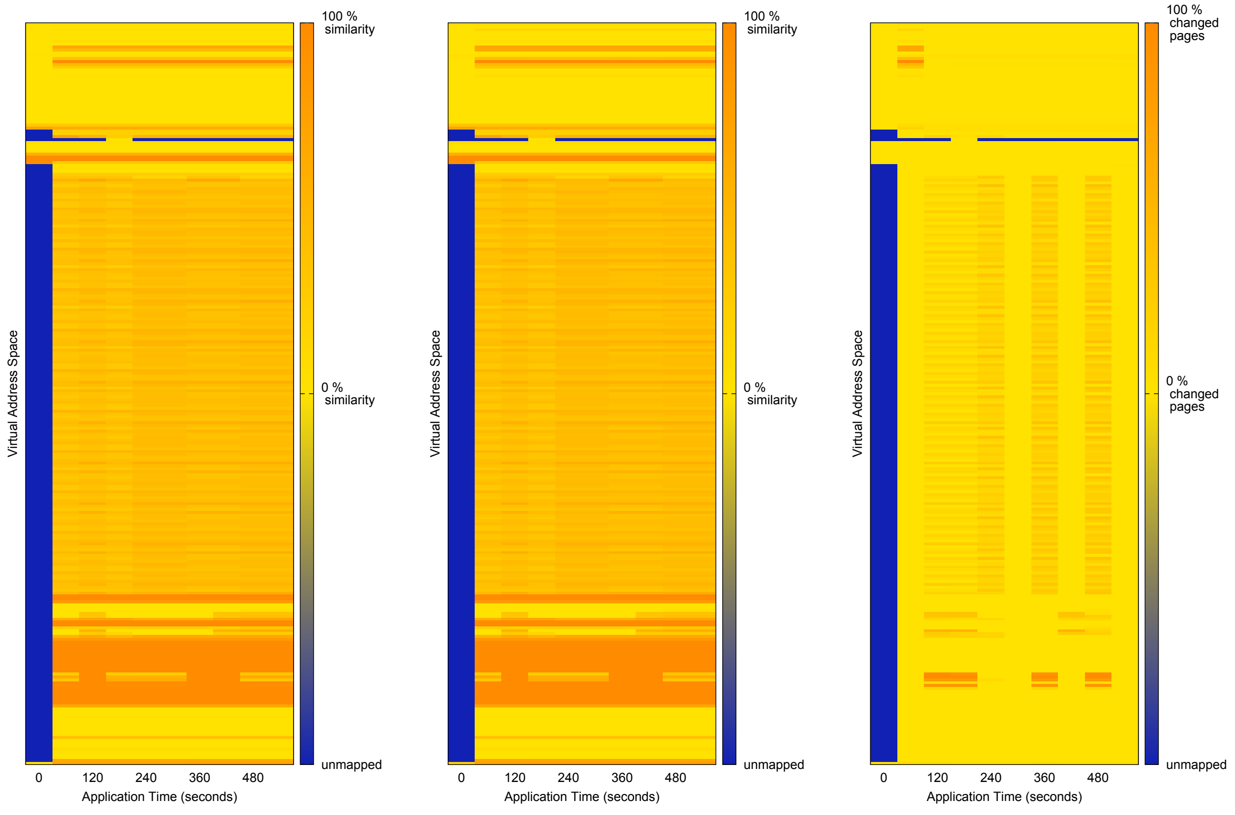
Heat Maps

This section includes heatmaps indicating both similarity and modification frequency for each of the eight applications that we considered.



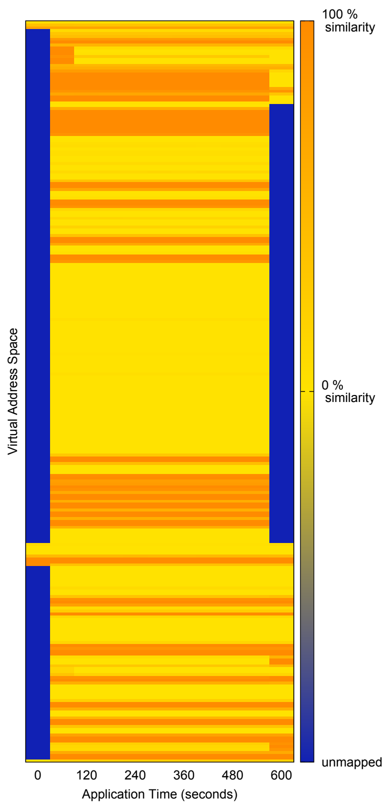
(a) Single Rank Heat Map **(b)** NUMA Domain Heat Map **(c)** Change Map

Figure 16. Address space behavior of AMG2006 as a function of application time.

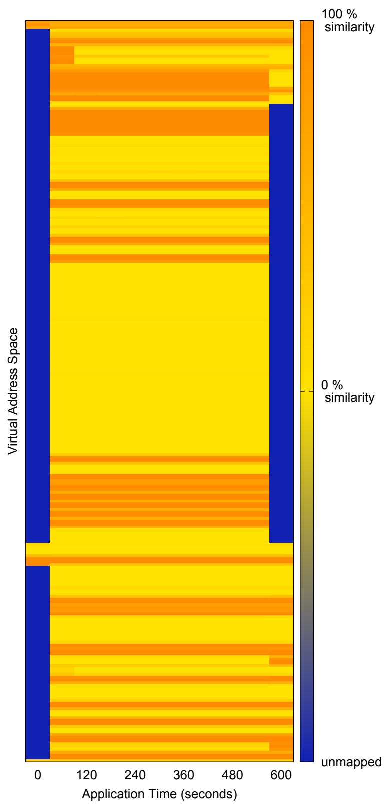


(a) Single Rank Heat Map **(b)** NUMA Domain Heat Map **(c)** Change Map

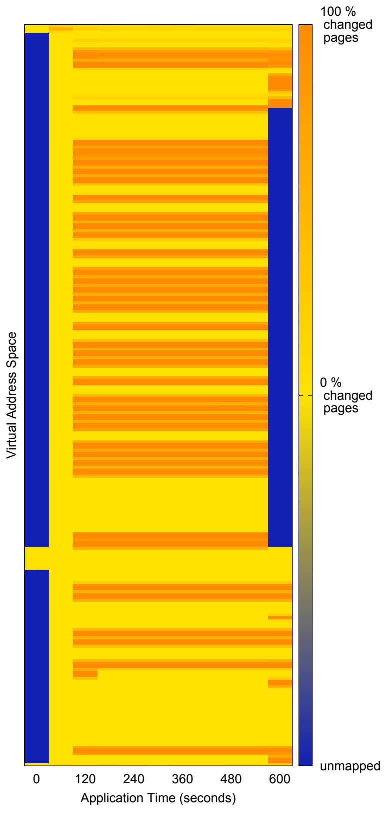
Figure 17. Address space behavior of CTH as a function of application time.



(a) Single Rank Heat Map

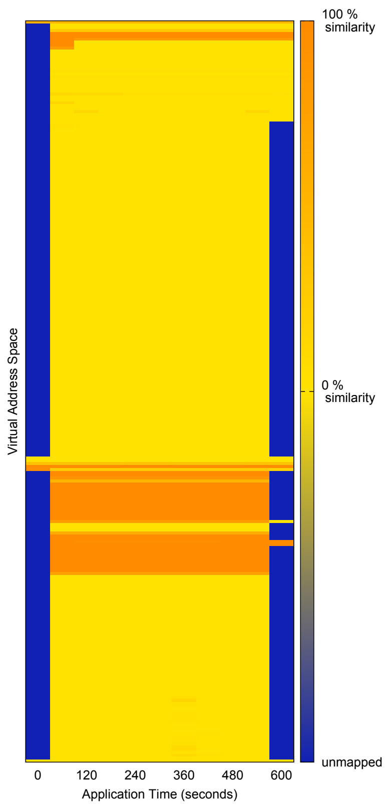


(b) NUMA Domain Heat Map

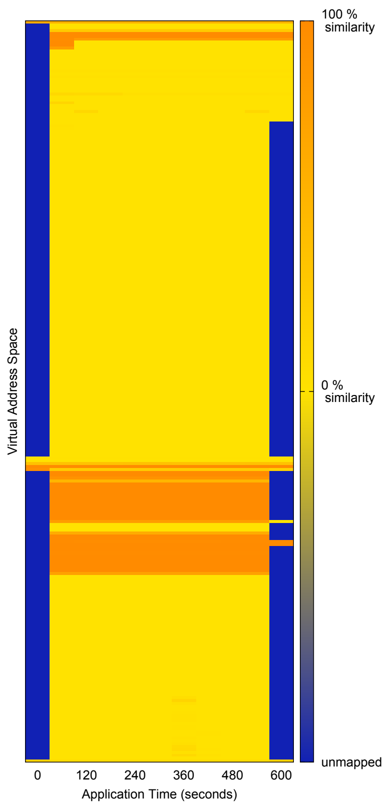


(c) Change Map

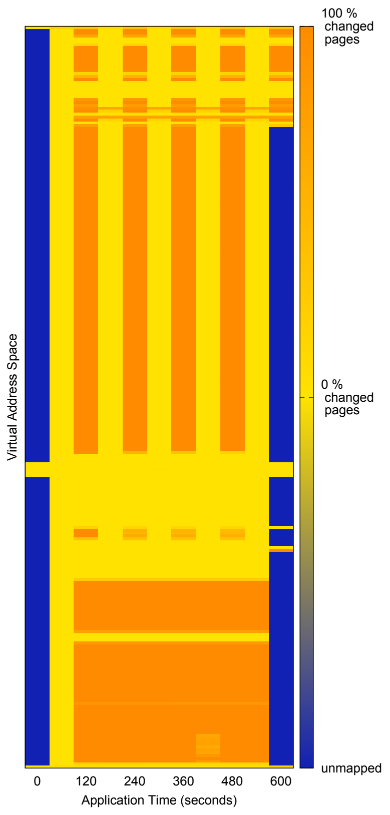
Figure 18. Address space behavior of IRS as a function of application time.



(a) Single Rank Heat Map

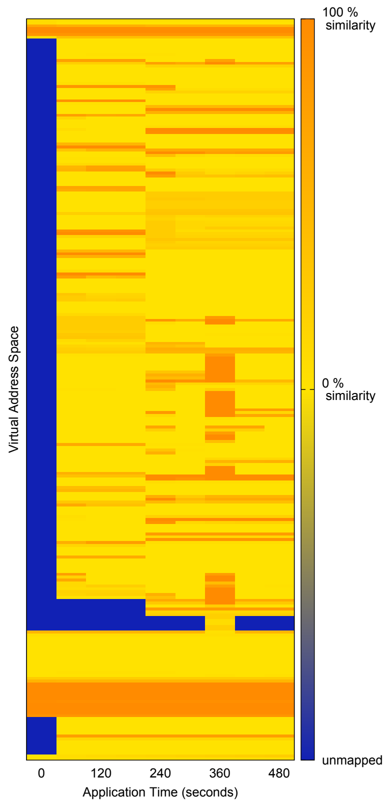


(b) NUMA Domain Heat Map

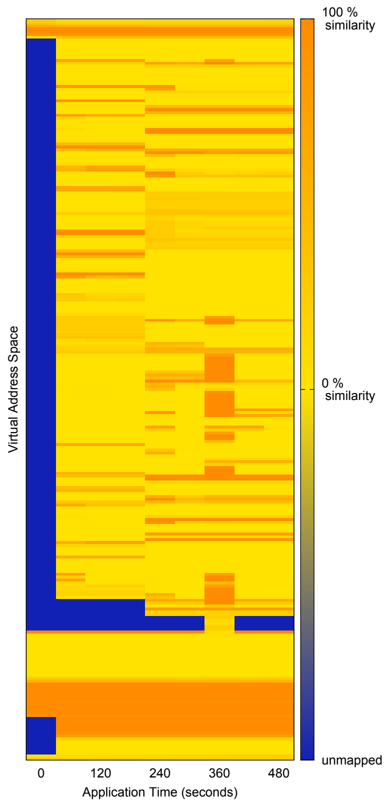


(c) Change Map

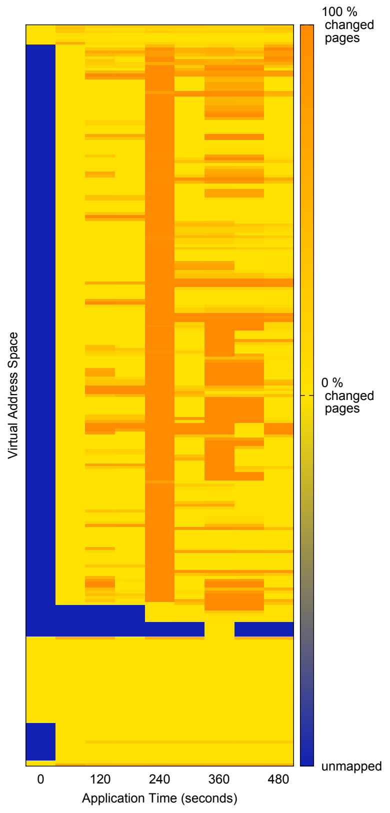
Figure 19. Address space behavior of LAMMPS as a function of application time.



(a) Single Rank Heat Map

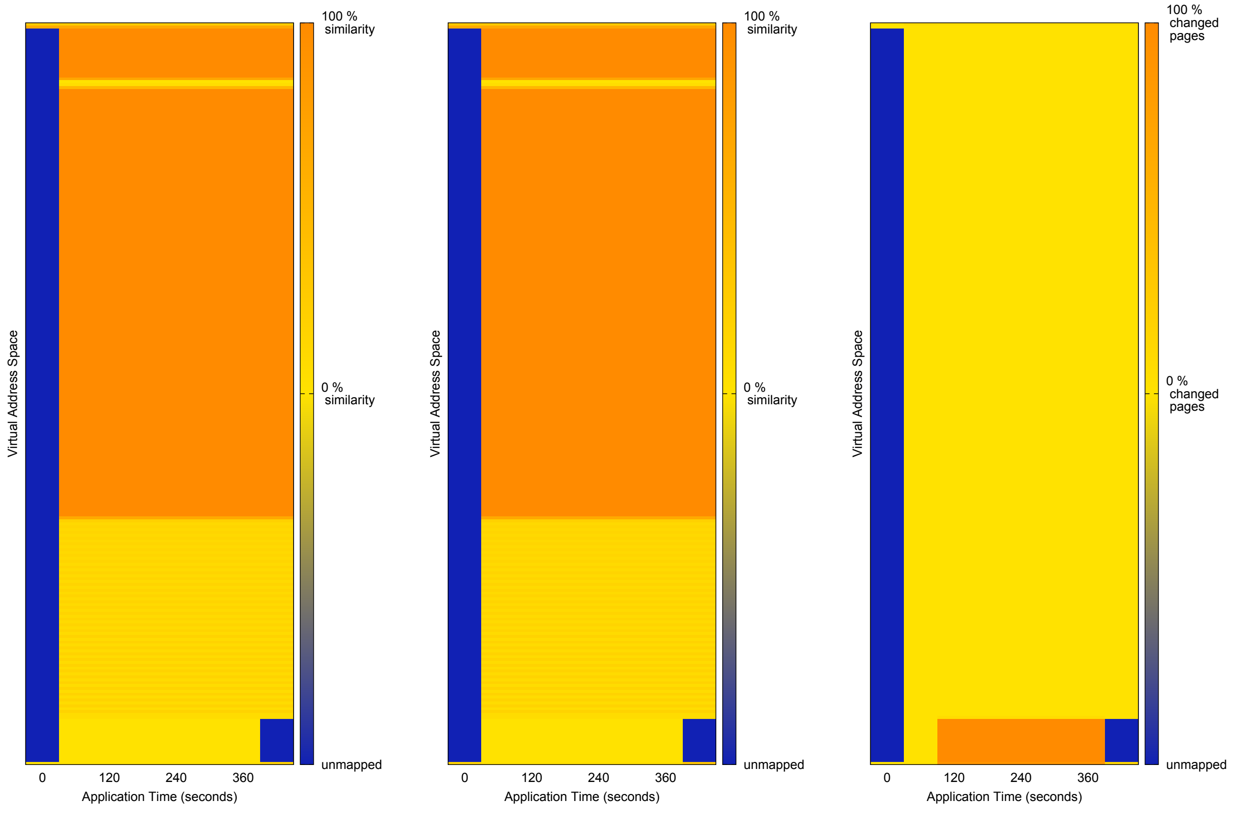


(b) NUMA Domain Heat Map



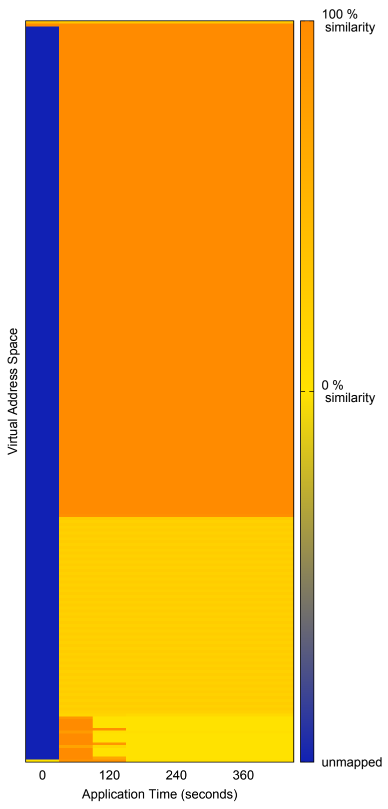
(c) Change Map

Figure 20. Address space behavior of SAMRAI as a function of application time.

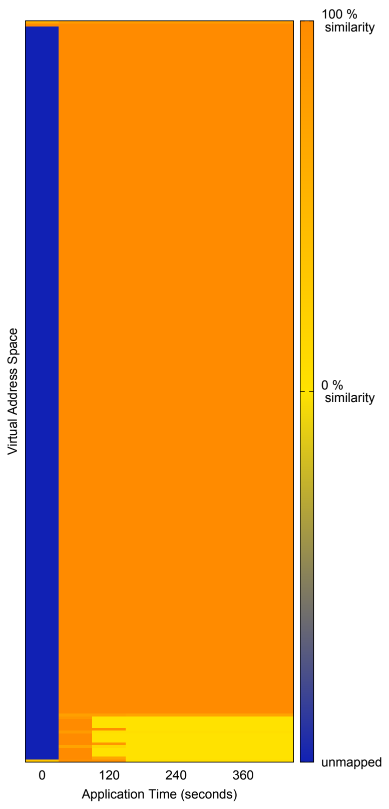


(a) Single Rank Heat Map **(b)** NUMA Domain Heat Map **(c)** Change Map

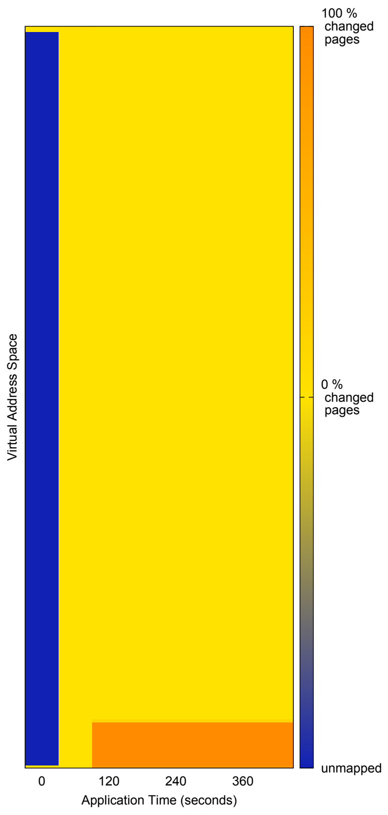
Figure 21. Address space behavior of Rank 0 running HPCCG as a function of application time.



(a) Single Rank Heat Map

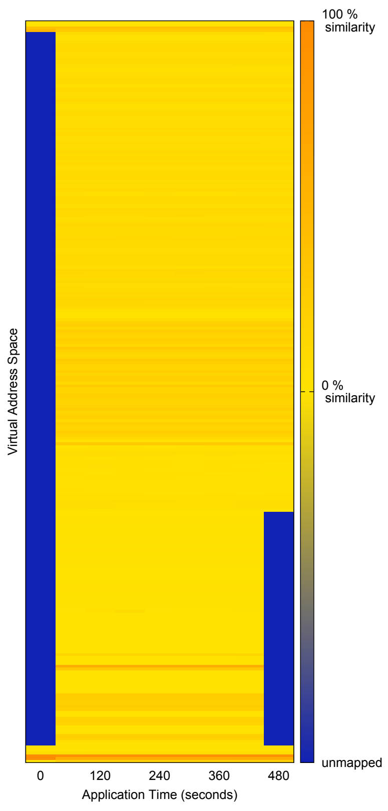


(b) NUMA Domain Heat Map

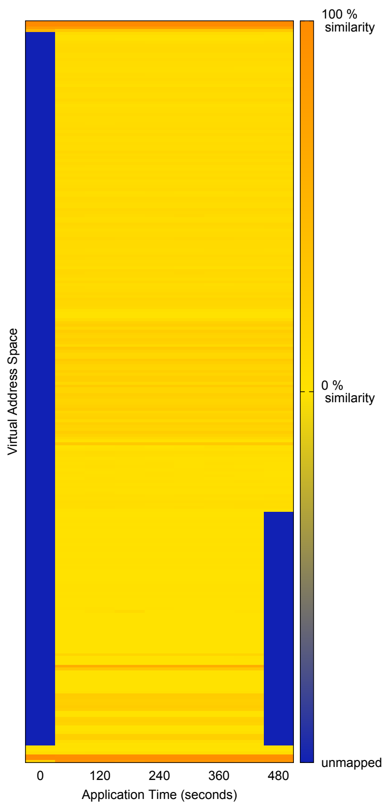


(c) Change Map

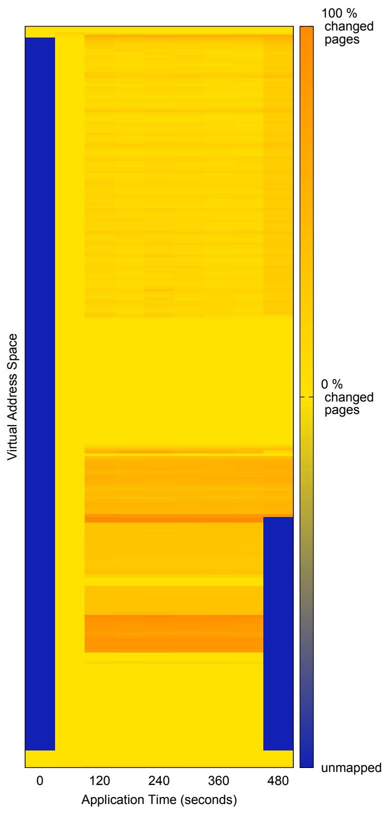
Figure 22. Address space behavior of Rank 1 running HPCCG as a function of application time.



(a) Single Rank Heat Map

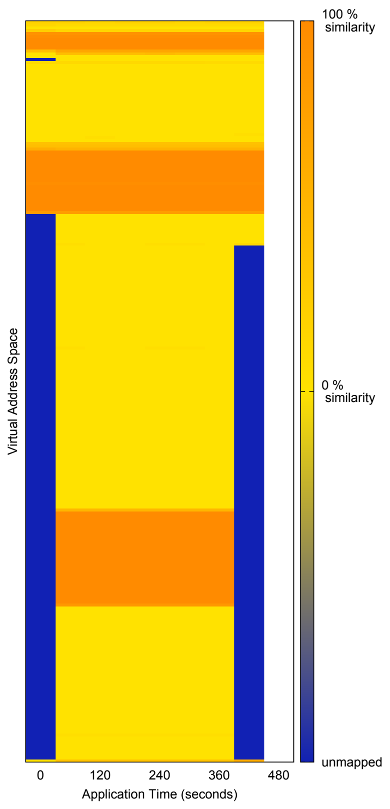


(b) NUMA Domain Heat Map

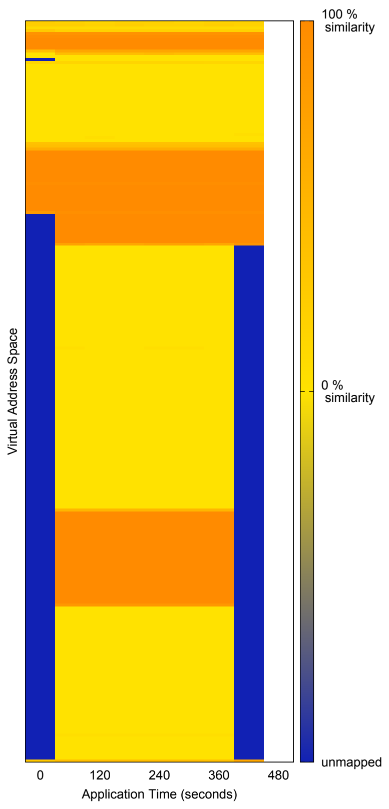


(c) Change Map

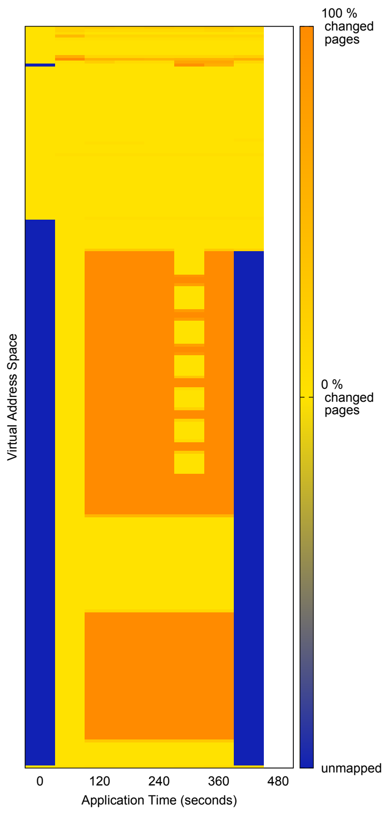
Figure 23. Address space behavior of phdMesh as a function of application time.



(a) Single Rank Heat Map



(b) NUMA Domain Heat Map

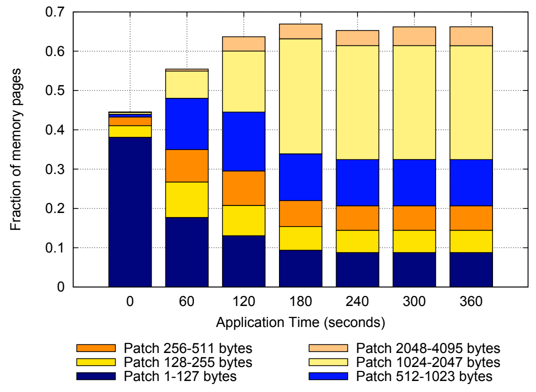


(c) Change Map

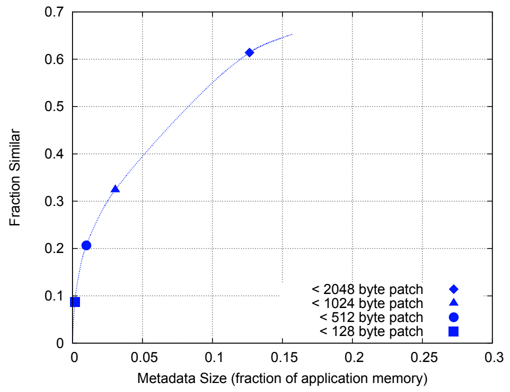
Figure 24. Address space behavior of Sweep3d as a function of application time.

Patch Size Distribution & Metadata Costs

This section includes plots illustrating the patch size distribution in the memory of Rank 0 and the associated metadata costs as a function of the size of application memory for each application over its lifetime. The metadata costs are computed based on the memory snapshot, excluding the initialization and finalization snapshots, of each application that contains the smallest fraction of similar and duplicate pages.

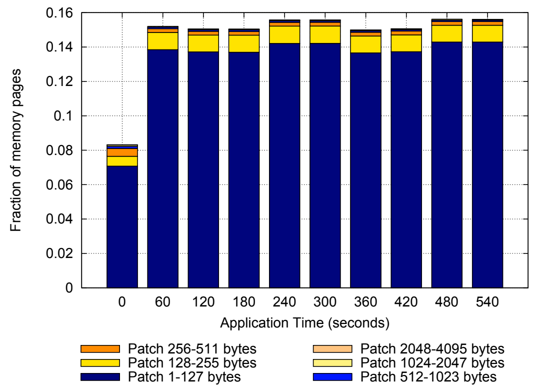


(a) Patch Size Distribution

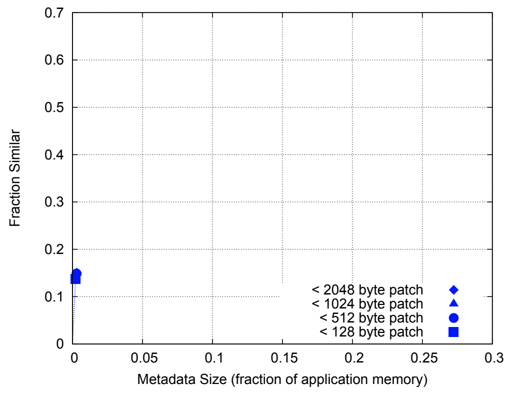


(b) Metadata Costs

Figure 25. Patch size data for AMG as a function of application time.

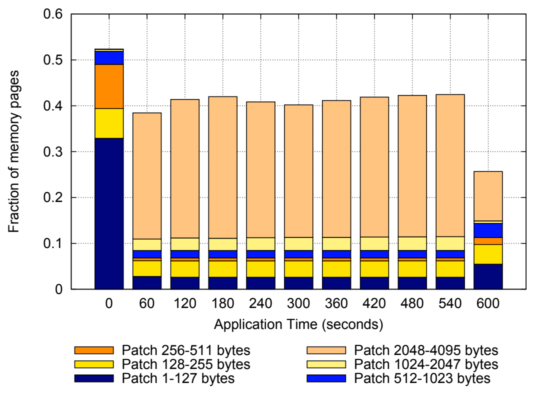


(a) Patch Size Distribution

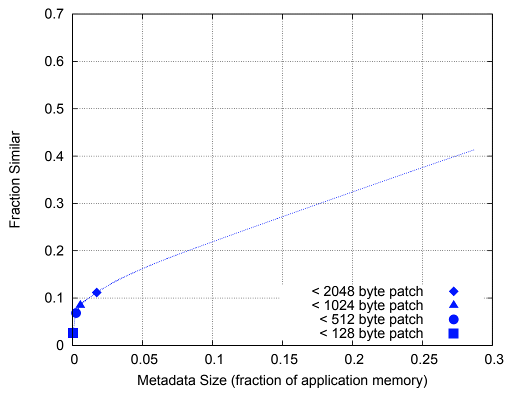


(b) Metadata Costs

Figure 26. Patch size data for CTH as a function of application time.

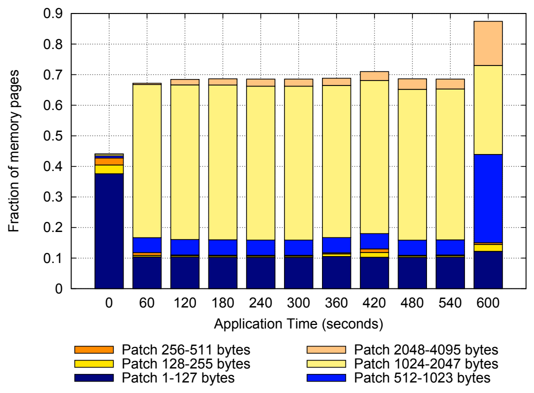


(a) Patch Size Distribution

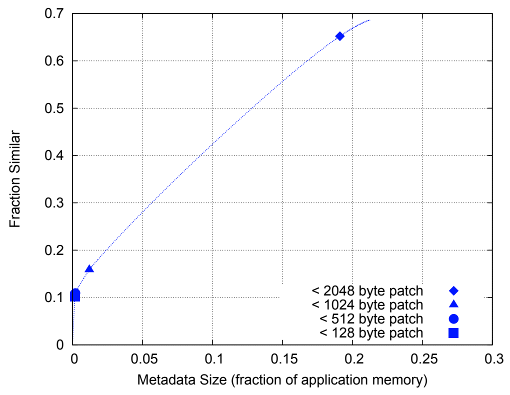


(b) Metadata Costs

Figure 27. Patch size data for IRS as a function of application time.



(a) Patch Size Distribution



(b) Metadata Costs

Figure 28. Patch size data for LAMMPS as a function of application time.

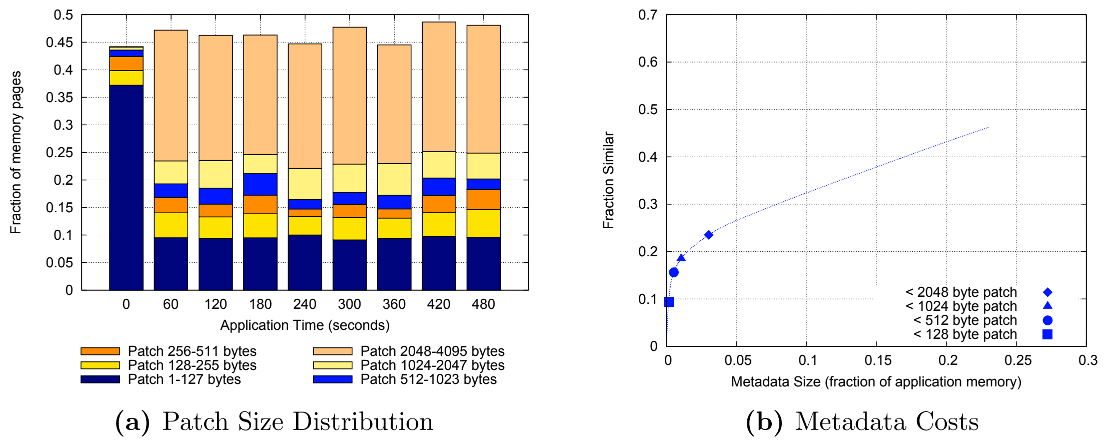


Figure 29. Patch size data for SAMRAI as a function of application time.

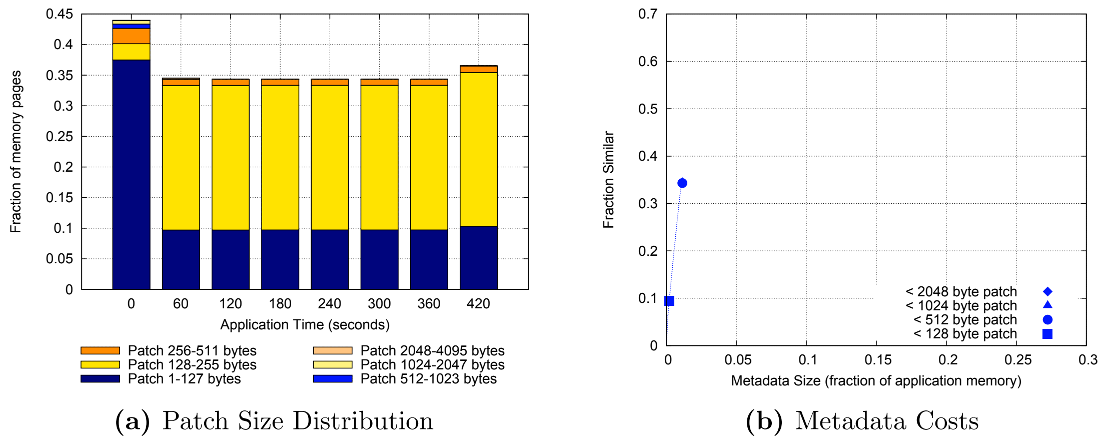
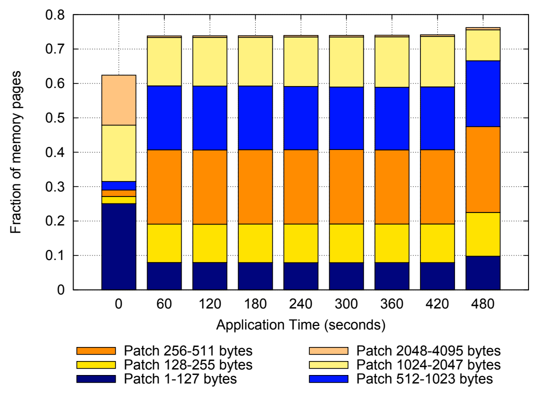
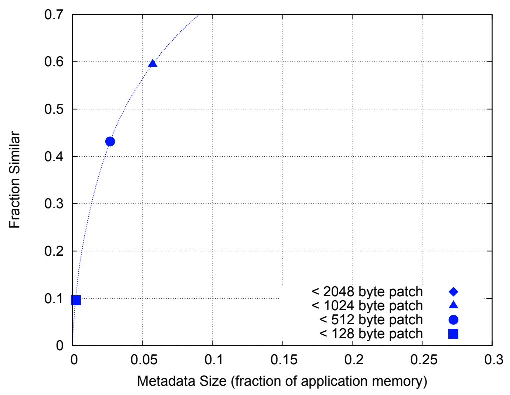


Figure 30. Patch size data for HPCCG as a function of application time.

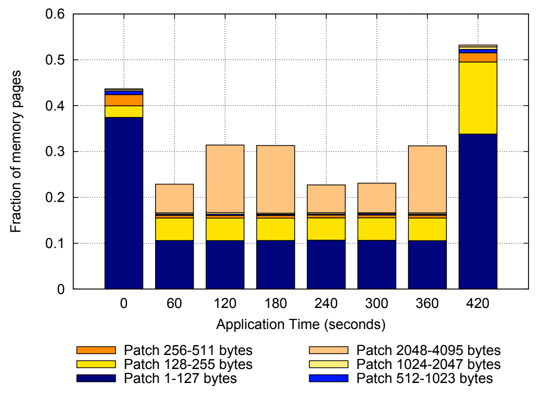


(a) Patch Size Distribution

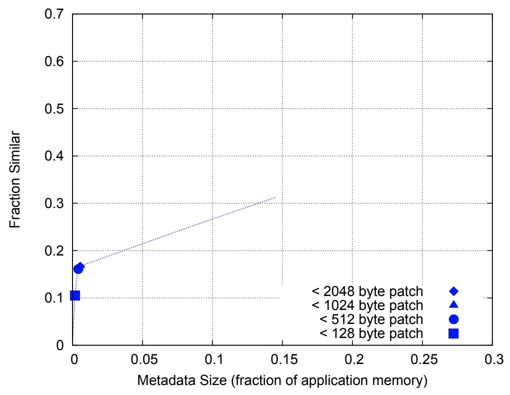


(b) Metadata Costs

Figure 31. Patch size data for phdMesh as a function of application time.



(a) Patch Size Distribution



(b) Metadata Costs

Figure 32. Patch size data for Sweep3D as a function of application time.

References

[1] Arcangeli, A., Eidus, I., and Wright, C. Increasing memory density by using KSM. In *Proceedings of the Linux Symposium, 2009, Montreal, Quebec* (2009), pp. 19–28.

[2] Biswas, S., Supinski, B. R. d., Schulz, M., Franklin, D., Sherwood, T., and Chong, F. T. Exploiting data similarity to reduce memory footprints. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium* (Washington, DC, USA, 2011), IPDPS '11, IEEE Computer Society, pp. 152–163.

[3] Bugnion, E., Devine, S., Govil, K., and Rosenblum, M. Disco: running commodity operating systems on scalable multiprocessors. *ACM Trans. Comput. Syst. 15*, 4 (Nov. 1997), 412–447.

[4] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A. Difference Engine: Harnessing memory redundancy in virtual machines. *Commun. ACM 53*, 10 (Oct. 2010), 85–93.

[5] Henson, V., and Yang, U. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics 41*, 1 (2002), 155–177.

[6] Heroux, M. A., Doerfler, D. W., Crozier, P. S., Willenbring, J. M., Edwards, H. C., Williams, A., Rajan, M., Keiter, E. R., Thornquist, H. K., and Numrich, R. W. Improving performance via mini-applications. Tech. Rep. SAND2009-5574, Sandia National Laboratory, 2009.

[7] Lawrence Livermore National Laboratories. IRS: Implicit Radiation Solver 1.4 Build Notes. `https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/irs/irs.readme.html`.

[8] Lawrence Livermore National Laboratories. SAMRAI. `https://computation.llnl.gov/casc/SAMRAI/index.html`.

[9] Lawrence Livermore National Laboratories. ASC Sequoia Benchmark Codes. `https://asc.llnl.gov/sequoia/benchmarks`, August 2009.

[10] Los Alamos National Laboratories. Sweep3d. `http://www.c3.lanl.gov/pal/software/sweep3d/sweep3d_readme.html`, 1999.

[11] McGlaun, J., Thompson, S., and Elrick, M. CTH: a three-dimensional shock wave physics code. *International Journal of Impact Engineering 10*, 1 (1990), 351–360.

[12] Quinlan, S., and Dorward, S. Venti: a new approach to archival storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies* (2002), vol. 4.

[13] Sandia National Laboratories. Mantevo. `http://software.sandia.gov/mantevo`.

[14] Sandia National Laboratories. The LAMMPS molecular dynamics simulator. `http://lammps.sandia.gov`, April 2010.

[15] Tuininga, A. cx_bsdiff. `http://starship.python.net/crew/atuining/cx_bsdiff/index.html`, February 2006.

[16] Vaughan, C., Rajan, M., Barrett, R., Doerfler, D., and Pedretti, K. Investigating the impact of the cielo cray xe6 architecture on scientific application codes. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on* (2011), IEEE, pp. 1831–1837.

[17] Waldspurger, C. A. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev. 36*, SI (Dec. 2002), 181–194.

[18] Xia, L., and Dinda, P. A. A case for tracking and exploiting inter-node and intra-node memory content sharing in virtualized large-scale parallel systems. In *Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing* (New York, NY, USA, 2012), VTDC '12, ACM, pp. 11–18.

[19] Yang, T., Jiang, H., Feng, D., Niu, Z., Zhou, K., and Wan, Y. DEBAR: A scalable high-performance de-duplication storage system for backup and archiving. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on* (2010), IEEE, pp. 1–12.

[20] Zhu, B., Li, K., and Patterson, H. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2008), FAST'08, USENIX Association, pp. 18:1–18:14.

DISTRIBUTION:

1	MS	1319	Kurt Ferreira, 1423
1	MS	1319	Scott Levy, 1423
1	MS	1322	Aidan Thompson, 1425
1	MS	1318	Christian Trott, 1426
1	MS	0899	Technical Library, 9536 (electronic copy)
1	MS	0359	D. Chavez, LDRD Office, 1911

Sandia National Laboratories