iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** V **Month of publication:** May 2017

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Memory Management For Heterogeneous Main Memory

Tejaswini R M[1]
[1]*Assistant Professor, Department of Computer Science and Engineering* GSSS Institute of Engineering and Technology for Women

Abstract: ***In the near future new technologies will make it possible to enlarge the main memory layer in the current memory hierarchy using devices with small cost and access penalties, such as the Storage Class Memory (SCM). In order to use those technologies as efficiently as possible, we need to understand how the developer and the operating system can get the best performances while managing a new heterogeneous main memory, which consists of multiple types of memory with different volumes and different access speeds. We found that the most reasonable way to introduce these new technologies into any usable memory system would be by using a new automated layer that selects the most appropriate memory levels for allocating space in the memory complex, and that moves data between memory levels of the memory complex for optimizing performance in the fashion of paging algorithms. Specifically, we discovered that this memory management is optimized using a modification of the Aging algorithm (a directive of the LRU concept) – a modification which can improve the access speed of the heterogeneous main memory by about 75%, and that manages to achieve the same or better Hit / Miss ratio in almost all cases in comparison to the current alternatives.***

Keywords-Memory Hierarchies, Main Memory, Paging, Aging Algorithm, Storage Class Memory, Heterogeneous System.

I. INTRODUCTION

Memory and storage are often assumed to be unsophisticated, 'flat' resources, with simple properties, such as a constant access time. Over the years this assumption has been proven to be wrong, and understanding of the memory hierarchy could be useful in order to enhance the performance of an algorithm or a data structure. For example, the Storage Class Memory (henceforth, SCM) is a new technology which represents a new hybrid form of storage and memory with unique characteristics, meaning a memory which is non-volatile, cheap in per bit cost, has fast access times for both read and writes using cache line access, and is solid state. Also, the SCM is supposed to have different versions with different access speeds and different volumes, meaning that it might be possible to add different SCM devices to the memory hierarchy as an extension of the RAM, and manage this enlarged heterogeneous main memory using special algorithms. Our hypothesis is that achieving appropriate transferability between these new heterogeneous main memory levels may be possible using ideas of algorithms employed in current virtual memory systems, and that an efficient memory-aware adaptation of those algorithms to a heterogeneous main memory is achievable [1].

In order to reach the conclusion that our hypothesis is correct, we investigated various paging algorithms, and found the ones that could be adapted successfully from a standard memory hierarchy to a hierarchy with heterogeneous main memory. We discovered that using a memory-aware adaptation of the Aging paging algorithm results in the best performances in terms of Hit / Miss ratio and access speed.

Linux-based OS/Rs have emerged as the dominant environment for many modern HPC systems [12], [13], [14] due to their support of extensive feature sets, ease of programmability, familiarity to application developers, and general ubiquity. Linux environments provide tangible benefits to both usability and maintainability, while generally offering acceptable performance in a dedicated and properly configured HPC system. However, we argue that as HPC systems continue to increase the degree of local workload consolidation, a commodity OS/R architecture is ill-suited to provide an appropriate level of performance for HPC class applications. This is because commodity systems, and Linux in particular, are designed to maximize a set of design goals that conflict with those required by HPC applications. Specifically, commodity systems are almost always designed to maximize resource utilization, ensure fairness, and most importantly, gracefully degrade in the face of increasing loads. These goals often directly conflict with those of HPC environments that are generally characterized as requiring consistent performance in the face of sustained heavy loads.

II. BACKGROUND

Operating systems (henceforth, OS) implement the virtual memory mechanism that extends the working space for applications, mapping an external memory file (page file) to virtual addresses. This idea supports the Random Access Machine model [2] in which a program has an infinitely large main memory. With virtual memory, the application does not know where its data is located, whether in the main memory or in the page file. This abstraction does not have large running time penalties for simple sequential access patterns: The OS is even able to predict them and to load the data ahead of time.

For more complicated patterns, especially in HighPerformance Computing (henceforth, HPC), these remedies are mostly not useful and even might be counterproductive. The page file is accessed very frequently, the executable code can be swapped out in favour of unnecessary data, and the page file is highly fragmented and thus many random I/O operations are needed for scanning. Therefore, in this scenario, there are two options to resolve the problem: the first option is increasing the Hit / Miss ratio and the access speed in the virtual memory mechanism, and the second option is an explicit handling of memory accesses. In this scenario, the applications and their underlying algorithms and data structures should care about the pattern and the number of memory accesses (I/Os), which they cause.

Several simple models have been introduced for designing I/O-efficient algorithms and data structures. The most realistic model is the Parallel Disk Model (PDM) of Vitter and Shriver [3]. In this model, I/Os are handled explicitly by the application. The most common implementation of the PDM model can be found at the STXXL project [4]. The core of STXXL is an implementation of the C++ standard template library STL for external memory (out-of-core) computations, i.e., STXXL implements containers and algorithms that can process huge volumes of data that only fit on disks. While the compatibility to the STL supports ease of use and compatibility with existing applications, another design priority is high performance. The performance features of STXXL include transparent support of multiple disks, variable block length, overlapping of I/O and computation, and prevention of OS file buffering overhead.

A. Future Memory: Storage Class Memory

Storage is considered to be a mechanical HDD that supplies virtually unlimited capacity when compared to RAM, and it is also perpetual, which means that data is not lost if the computer happens to crash or disconnect from electricity. The issue with hard drives is that in various situations they are unable to supply data to applications with the sufficient speed, because of their mechanism and access fashion [1].

Storage Class Memory (SCM) [5] proposes to minimize or even close the widening gap between CPU processing speeds, the need to rapidly transfer big data blocks, and the read-write speeds suggested by HDD reliant systems. The SCM, widely known as Persistent Memory, is a technology which represents a new hybrid form of storage and memory with unique characteristics, meaning a memory which is non-volatile, cheap in a per bit cost, has fast access times for both read and writes using cache line access, and is solid state [6]. Also, the SCM is supposed to have different versions with different access speeds and different volumes, meaning that it may be possible to add different SCM devices to the memory hierarchy as an extension of the RAM, and manage this enlarged main memory using special algorithms, as the PDM model manages the disk complex using STXXL library.

The SCM has a unique mechanism. Created out of flashbased NAND, SCM is a new form of storage that can provide a middle step between high-performance RAM and cost-effective HDDs. It may very well provide read performance analogous to RAM (perhaps even better in some cases), and write performances that are significantly faster than HDD technology (factors of hundreds better than HDD and even beyond). Also, it is predicted that the production costs of SCM and HDD will be broadly similar by the end of this decade [7].

These new SCM devices connect to memory slots in a server and are mapped and accessed in the same fashion as the memory, even though they are slightly slower, and they can be addressed atomically at either the byte or the block level, unlike previous eras of storage technology. The SCM can be used directly as execution memory or data storage memory. The current SCM products include the improved Flash [8], the Phase Change Memory (PCM) [9], the Magnetic RAM (MRAM) [10], the Solid Electrolyte RAM – Nano-Ionic RAM [11], the Ferroelectric RAM (FRAM) [12] and the Memristor [13].

OS are likely to use the SCM as either very fast block storage devices formatted by file systems and databases, or as direct memory mapped “files” for next generation of programs. In the near future the SCM is predicted to modify the form of programs, the access form to storage, and the way that storage devices themselves are built. Therefore, a combination between SCM technology and the existing RAM, using a new memory allocation manager (henceforth, MAM) that will act like STXXL – but using cache line access fashion – will be likely to achieve a new level of performance for memory-aware data structures.

B. Data Structures Problem in Usage of the Memory Allocation Manager

There are clear benefits of using a MAM at the design and implementation stage of any data structure which needs to handle massive data sets. For example, given that the keys are arranged in a list form, there is an option that the first immediate key will be stored in the fastest memory available, and as far as the list expands, the other keys will be stored in slower memories.

Therefore, it seems reasonable to re-modify the data structures to use those kinds of paradigms using a MAM, as in Fig. 1. However, this kind of solution arises a problem, which is an inherent part of this solution itself. The reason for this problem is that a MAM require not only remodifying the memory platform and the access to it, but obviously also to re-write the memory-oblivious codes that is currently based on transparent memory access. The chances that such a significant re-write would be implemented in current codes using HPC platforms – the primary target group which need this massive enlargement of the main memory – is quite low. Most of the centers which use large computer clusters have programs that are intended to be operational for ages and are very big, complex and sustainable. Hence, the solution of rewriting the whole code and redefining the data structures to use a MAM is not possible nor plausible, and can be efficient only for several specific applications that are written today, yet not for past applications.

III. PAGE REPLACEMENT ALGORITHMS IMPLEMENTATION FOR HETEROGENEOUS MAIN MEMORY

Computers often have five memory levels with different properties [1]. Three of these levels dwell on the processor chip, one level is the RAM memory and one level is the storage memory, such as SSD or HDD. The levels on the processor chip are referred to as L1 cache, L2 cache and L3 cache. L1 cache is a rather small piece of memory with extremely high access time, used directly by the processor. L2 cache is slightly slower and vastly larger than L1 cache. L3 cache is slower than L2 and L1 caches, and it is shared by all cores. Also, it is worth mentioning that there are architectures with an additional cache memory level, L4, which may result better performance than sole enlargement of the L3 cache level [14].

However, because the amount of data that maps to a cache section is generally much larger than the associativity of the cache, a designated replacement policy necessarily needs to determine which data to evict when a cache miss occurs [1]. Unfortunately, sufficiently precise documentation of the specific logical organization of the memory hierarchy is seldom available publicly, and the current knowledge on the different cache management policies is base on specific reverse engineering simulations [15]. Nonetheless, the current page replacement algorithms concepts are well known [16]. Replacement policies try to identify which data can be a proper candidate for eviction, by basing their resolutions on the chronology of the memory accesses. Distinguish replacement policies of this kind are Least Recently Used (LRU) and Least Frequently Used (LFU); pseudo-LRU (PLRU), an efficient variant of LRU (like the Aging policy); First-In First-Out (FIFO), also known as Round Robin; and Not-Recently Used (NRU) [16].

A. Usage of Paging Concept in a Heterogeneous Main Memory

Our hypothesis is that achieving a transferability between memory levels may be possible using ideas of algorithms employed in current virtual memory system, and that the adaptation of those algorithms from a standard memory hierarchy to a heterogeneous main memory may be possible. In that notion, each virtual memory page is a data entry (or entries), and each of the virtual memory swaps is done as a MAM would do if it would have supplied with knowledge of the appropriate levels. However, our approach is different from that of the paging algorithms used for cache hierarchy, specifically, cache algorithms simply transfer data to an immediate lower memory level when the current memory level is full, till the data reach to the main memory or evicts to the storage. In contrast, our management algorithms for heterogeneous main memory will allow data to flow from one level to another freely, and not only the way down in the hierarchy or directly to the most upper level when referenced. Also, because the heterogeneous main memory handles data that is not urgently needed as data which remains in the cache, there is no need to use the same algorithmic simplicity of the cache mechanism, which simply evicts pages to the next lower level when the current level reaches its capacity. Instead, it is plausible to evict the data to some specific memory level based on extra knowledge that the OS already has. This approach, however, is not in use in the cache management algorithms because of the time overhead that cannot be tolerated in the upper levels of the memory which reside near the CPU [16].

In order to reach a conclusion that our hypothesis is correct, we clarified which of the paging algorithms can be adapted successfully from a standard memory hierarchy to a heterogeneous main memory using the ideas above, and after thoroughly investigating the current paging mechanism and the main paging algorithms [18], we found the LRUNFU algorithms – and specifically their Aging derivative – is the best match to our goals.

B. The Memory-Aware Aging Page Replacement Algorithm for Heterogeneous Main Memory

The Aging algorithm is a modification of NFU algorithm which makes it possible to simulate LRU algorithm quite well. Instead of only incrementing the counters of pages referenced, the variation has two parts: First, the counters are shifted right once before the R bit is inserted, i.e., there is actually a division by 2 of the represented decimal number. Second, the R bit is inserted to the leftmost bit, instead of inserting it to the rightmost bit.be a memory-aware algorithm, for usage in a heterogeneous main memory model, can even add another level of sophistication, especially because of the existence of a linear proportion between the degradation of the referenced bits and the amount of time that a specific page has not been in use. In this hypothesis there is an interesting phenomenon; specifically, there is a possibility to create a direct link between the amount of zeroes in the beginning of the page referenced bits to the level of memory that that page should be evicted to according to its usage proportion (L in Formula (1)). This is achieved using a calculation which should take only few floating-point operations, and which is based on information that the OS already holds. Based on the knowledge that the amount of zeroes points to the amount of unreferenced past clock ticks – and therefore on the page aging status

A life cycle of a page should be as in the following route: First, the page is inserted into the memory hierarchy (using Insertion function below; Algorithm 1); then, depending on its aging status, it is 'diffusing' to lower memory levels in the complex hierarchy (using Update function below; Algorithm 2). It is worth noting that if the page is being referenced, it is redirected right to the first level (also using the Update function). Hence, by forming a dynamic pyramid hierarchy of both page and memory necessity it becomes possible to get significantly better performances for the Aging paging algorithm in a heterogeneous main memory.

IV. CONCLUSIONS

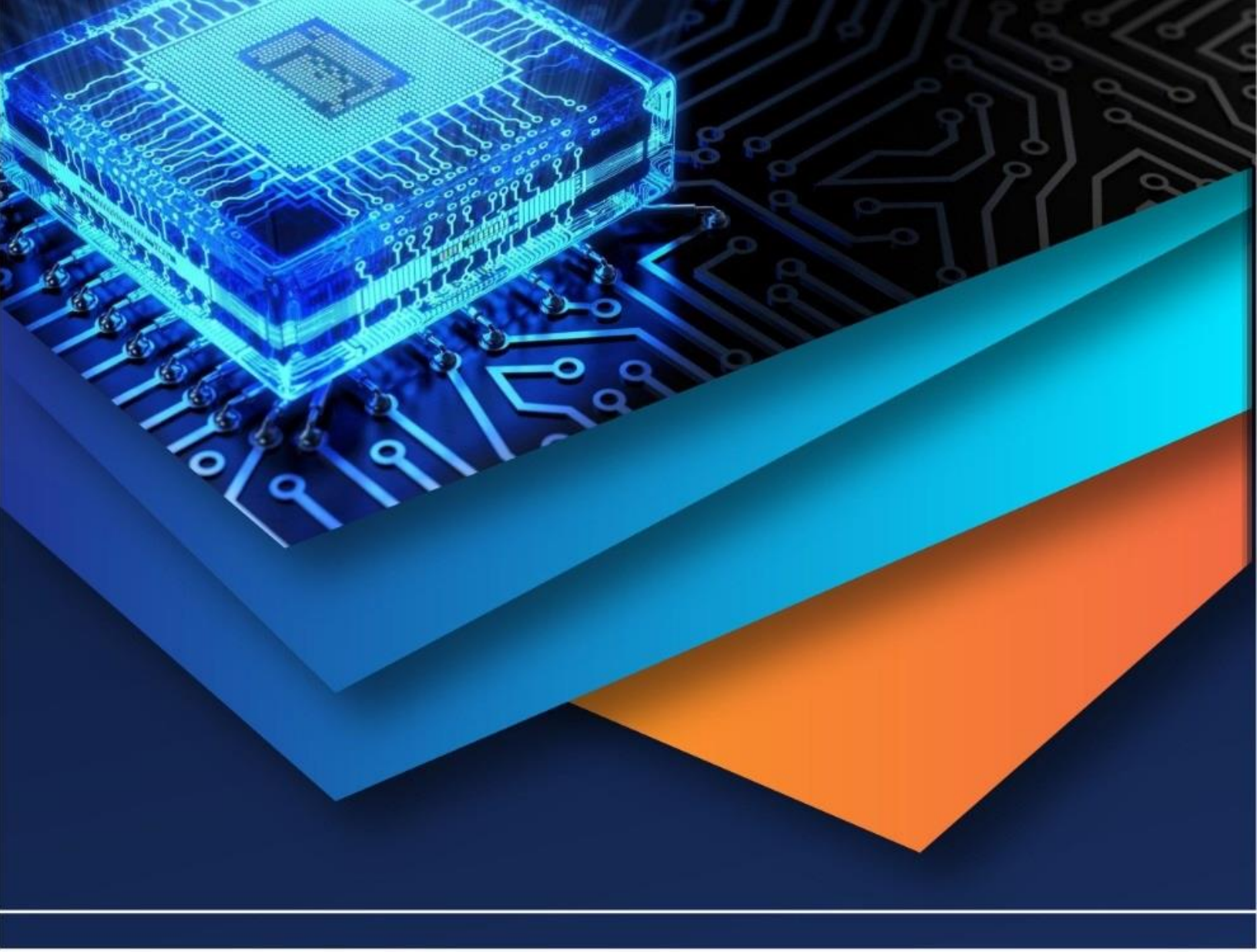
Those benchmarks, results and analysis lead to the conclusion that it would be beneficial to use the memory aware Aging paging algorithm in a heterogeneous main memory which includes SCM devices in standard computing systems as well as in HPC clusters. This paper opens a number of prospective directions for future research. One immediate direction is to explore how the memory-aware Aging paging algorithm is reacting when the memory levels are not from the same class, and what exactly does that mean in aspects of cost, volume, memory access fashion and access of speed. Another direction is to understand how to optimize other memory management algorithms which are not paging algorithms for beneficial usage of the heterogeneous main memory. Finally, we also expect that in the near future the SCM invention will be a real and widespread technology, meaning that investigating actual SCM devices, applying our algorithm in managing them as part of a heterogeneous main memory, and comparing the results to the presented simulations would be a fertile ground for further research and development.

REFERENCES

[1] Andrew S. Tanenbaum, Modern Operating Systems, Prentice Hall, 4nd edition, 2014.
[2] Von Neumann, John, First Draft of a Report on the EDVAC, IEEE Annals of the History of Computing 4, 1993, 27-75.
[3] Vitter, Jeffrey Scott, Elizabeth A. M. Shriver, Algorithms for Parallel Memory, II: Hierarchical Multilevel Memories, Algorithmica 12.2-3, 1994, 148-169.
[4] Dementiev, Roman, Lutz Kettner, Peter Sanders, STXXL: Standard Template Library for XXL Data Sets, Softw., Pract. Exper. 38.6, 2008, 589-637.
[5] Freitas, Richard, Winfried Wilcke, Bülent Kurdi, G. W. Burr, Storage Class Memory, Technology and Use, In Tutorial, 6th USENIX Conference on File and Storage Technologies, 2008.
[6] Huang, C.Y., Storage Class Memory, Final Report – IEE5009: Memory Systems, Institute of Electronics, National Chiao-Tung University, Fall 2012.
[7] Hession David, Nigel Mc Kelvey, Kevin Curran, Storage Class Memory, International Journal of E-Business Development IJED 4.1, 2013.
[8] Burr, G.W., Virwani, K., Shenoy, R.S., Padilla, A., BrightSky, M., Joseph, E.A., Lofaro, M., Kellock, A.J., King, R.S., Nguyen, K. and Bowers, A, Large-scale (512kbit) integration of multilayer-ready access-devices based on mixed-ionic-electronic-conduction (MIEC) at 100% yield, In VLSI Technology (VLSIT), Symposium on (pp. 41-42), IEEE, 2012.
[9] Numonyx, The basics of phase change memory (PCM) technology, White Paper, 2010
[10] Shenoy, R.S., Gopalakrishnan, K., Jackson, B., Virwani, K., Burr, G.W., Rettner, C.T., Padilla, A., Bethune, D.S., Shelby, R.M., Kellock, A.J. and Breitwisch, M., Endurance and scaling trends of novel access-devices for multi-layer crosspoint-memory based on mixed-ionic-electronic-conduction (MIEC) materials, In VLSI Technology (VLSIT), 2011 Symposium on (pp. 94-95), IEEE, June 2011. [11] Byrne, S, University develops PMC memory, a potential Flash killer, myce, 1 November 2007.
[11] Freitas, Richard F., Winfried W. Wilcke., Storage-class memory: The next storage system technology, IBM Journal of Research and Development 52.4.5: 439-447, 2008
[12] Chen, Y.C., Rettner, C.T., Raoux, S., Burr, G.W., Chen, S.H., Shelby, R.M., Salinga, M., Risk, W.P., Happ, T.D., McClelland, G.M. and Breitwisch, M., Ultra-thin phase-change bridge memory device using GeSb, In International Electron Devices Meeting (pp. 777- 780), December 2006.
[13] Bezerra, Carlos Eduardo B., Cláudio FR Geyer, A short study of the addition of an L4 cache memory with interleaved cache hierarchy to multicore architectures, Institudo de Informatica, Universidade Federal do Rio Grande do Sul
[14] Abel, Andrew, Jan Reineke, Reverse engineering of cache replacement policies in Intel microprocessors and their evaluation, Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. IEEE, 2014.
[15] Jacob, Bruce, Spencer Ng, David Wang, Memory systems: cache, RAM, disk, Morgan Kaufmann, 2010.

Volume 5 Issue V, May 2017
ISSN: 2321-9653

[16] Zheng, Ying, Brian T. Davis, Matthew Jordan, Performance evaluation of exclusive cache hierarchies, In Performance Analysis of Systems and Software, 2004 IEEE International Symposium onISPASS, pp. 89-96. IEEE, 2004.
[17] Majo, Zoltan, Thomas R. Gross, Memory management in NUMA multicore systems: trapped between cache contention and interconnect overhead, In ACM SIGPLAN Notices, vol. 46, no. 11, pp. 11-20. ACM, 2011.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY