Managing Scratchpad Memory Architecture for Lower Power Consumption Using Programming Techniques

Kavita Tabbassum[1], Shahnawaz Talpur[2], Noor-u-Zaman Laghari[3]

[1]Information Technology Centre, Sindh Agricultural University, Tando Jam, Hyderabad, Sindh, **PAKISTAN**
[2,3]Department of Computer Systems Engineering, Mehran University of Engineering & Technology, Indus Hwy, Jamshoro, Sindh, **PAKISTAN**

*Corresponding Email: kavita@sau.edu.pk

ABSTRACT

In embedded systems Scratch memory is generally used as an addition to caches or as a substitute of cache (Banakar et al., 2002), but due to their comprehensive ease of programmability cache containing architectures are still to be chosen in numerous applications. Power consumption of ported applications can be significantly lowers as well as portability of scratchpad architectures will be advances with our suggested language agnostic software management method. To enhance the memory configuration on relevant architectures, variety of present methods is reviewed for finding the chances of optimizations and usage of new methods as well as their applicability to numerous memory schemes are discussed in this paper.

Key Words: Scratch, Scratchpad, Memory Architectures

Source of Support: None, **No Conflict of Interest:** Declared

INTRODUCTION

During application execution scratch memory usually used for momentary storage of data and represents a category of local high speed storage. Using direct memory access instructions among main memory and scratchpad locations data are commonly transferred as in the hardware coherence approaches of maximum caches in contrast to being copied. In possibly definite potential (in the situation of hard real time applications) (Shivaraj and Dharishini, 2015), on-chip area savings and decreased energy usage are the distinctive supports, presented through scratchpad memory involved. In general purpose computing these attributes are also beneficial, however explicit data movement and the load of memory conscious program presents a significant task to current compilers and fails adoption through mainstream designs (Pena and Balaji, 2014). By identifying initiations and enhancement as well as merging a range of current methodologies, for managing scratchpad memory in a favorable ways by using software that describe a system is discussed in this paper.

To maximize usability the suggested scheme will follow an elementary set of requirements. The purpose of this research is to make the most of additional criteria and need that a suitable system selects programmability including data locality, hardware-aware execution drive efficiency and general presentation when probable. Intermediate representations are required to provide back for a set of functionality compatible with compiler intermediate and shared languages.

LITERATURE REVIEW

Scratch memory with hard-ware implementations have an extensive history. The IBM Cell architecture and multicore Digital Signal Processor (DSP) chips are the recent examples of interest. In a race to advance usability of these types of architectures recently different software memory management approaches have been constructed. At runtime transfer and storage of the frequently come across forms of memory objects are of primary concern. To load and run a generic procedure from the perspective of the required functionality, different approaches of immediate interest will be discussed in this research.

Static Allocation of Memory

On scratchpad architectures static memory distribution of code is easily assumed. Size requirements of global data and fixed-size text code may statically be computed by compilers and with sufficient available capacity into a scratchpad memory these can be trivially loaded. With symmetric access to main memory (Verma et al., 2005) and by including an instruction cache some architecture further simplifies this process. During the existence of dynamic contention or constraints on vacant memory space the job of static object apportionment turn out to be further puzzling, but supporting overlays and linkage editing can be mostly handled by loaders. Code may be limited to pointer values and load locality free addresses depending on accessibility of virtual memory addressing modes and multitasking capabilities.

By observing simple metrics the static distribution procedure can be enhanced, such as frequency of access, sizes of data types accessed, control movement features of the adjacent code and accordingly arranging distribution tasks. Comparative to the distinctive size of quick subroutines this is probable and only firmly required on the structural design with strictly restricted vacant memory size.

Automatic Allocation of Memory

During application execution of scratch memory comprising systems the stack utilization is properly maintained. Runtime allocation of a function frame is the mutual sample of this sort of memory utilization. In the local memory of an executing application, data assigned in this style need to be accessible whenever required. Pointer arithmetic and simple stack variable accesses are data dependencies that may include but must not be statically resolvable. While retargeting applications to scratchpad designs in assuring performance portability, consequently efficient management of these data is very important.

Using software managed cache rule called as "Comprehensive Globular Stack Organization" (Kannan et al., 2009) it was an initial effort to manage the stack. For insertion overlapping this arrangement identified API functions for every function request in an application, stated in the earlier unit just look like overlays. Inside a fixed-size memory the system permits automatic allocation according to instantaneous application and availability requirements by loading frames from main memory and through ejecting stack structures from resident scratch memory.

Later in an innovative arrangement named as "Smooth Stack Data Organization" (Lu et al., 2013) certain restrictions of the first method were mentioned. To the complete stack space this technique lengthens the organization granularity instead of individually at each function level. By providing automated pointer management (giving functions for write back and loading among the places and at the phase of definition through transforming local addresses into global addresses), to practice a linear queue structure that make the library simple, and to computerize the attachment of API calls by relating an acquisitive algorithm. Through adjusting the location of API calls the insertion algorithm targets towards moderate organization overhead, to the ideal cutting of a particular application's weighted call graph, formalized as applying a fixed-size constraint (according to the required stack size with weights statically assigned, for each function a directed graph containing nodes, and weighted by call count, edges for each function call). By indicating that their algorithm reaches the ideal outcome as well as for managing the function call placement, the authors deliver the formulation of Integer Linear Programming (ILP) but leaves the estimation of values and the creation of the weighted call graph as an open problem. With nontrivial control flow actual weights of programs might not be calculated statically.

Dynamic Memory Allocation

As with automatic distribution mention above, at runtime dynamic distribution of memory items is maintained, but related to data for example heap variables that may demand a large quantity of space that is possibly indeterminable ahead of time. Through system calls languages similar to C offer dynamic allocation services used for automatic allocation rather than demanding the API call insertion process, these function calls also needs the suitable properties of interpreting distribution demands.

For the programmed heap data management one of the present approaches is known as "Completely Automatic Heap Data Organization"(Bai and Shrivastava, 2000). As per local memory accessibility by authors this method was demonstrated clearly for maintaining pointer transformations among global and local address spaces by means of an implementation that improved with additional API functions in GCC 4.1.1 the _malloc and _free functions. By means of a system of non-standard virtual memory simulation this pointer alteration can be automatically assumed, note that by the same authors in continuation to work in its replacement policy, by preventing the associativity of the software managed heap cache data, the configuration runtime overhead was reduced for newly ejected memory objects (Bai, 2014) by the addition of a "\victim buffer". Conversely entire constraints were depend on application, but the architecture associating SIMD comparison instructions was stated to be effective to accelerate tag operations.

METHODS

Scratch memory with a two-part structure after fulfilling the claimed requirements may be considered through a comprehensive generalize tactic for software management. Rules for program executions are created initially in a functionally precise manner by using a sequence of demonstration. Later, working on one or more of these representations following a sequence of conversions are defined.

Assumptions can be validated into simplify subsets of execution logic or others are clear in order to de obfuscate valuable belongings; the data dependence graph and the static control flow are the other included common representations. This method is analogous to that applied in most of the compilers that depend on a well-defined intermediary representation with the exception of applications comprising the hand-tuned assembly level code. Rather than code having expected degraded meaning as an outcome during previous or else suboptimal manual optimization struggles, as compiler optimizations remain towards progress it is expected that the greatest efforts for the program writer is to develop as well as to be focus on condition that give flawless commands to a proficient tool chain. As thread numbering rise or as thread performance deviates so is the capability of the program writer to compose the equivalent code restricted also via their skill to grip synchronization, similarly area constraints for a limited code distance dynamic optimization methods applied in hardware are also restricted.

Scratchpad Memory Allocation

In two categories scratch memory applications repeatedly divided: as a complementary storage component that can be incorporated in memory or as an individual vacant local memory stock they can be built in architecture. In the memory hierarchy they may be appear at one or more levels and access constraints or follow a set of privilege that may be globally accessible whether functionally or physically imposed. Scratchpads merely support elementary modes of physical addressing only; same as in shared embedded structures it may support virtual memory. Which particular approaches of memory distribution and management will produce the maximum advantage; deprived of emulation of lost functionality of operating systems it supports; at a high level all of these variables jointly determine the activities of the defined memory scheme.

By concentrating on associated memory designs process, from the above segments it is concluded that the proposed memory allocation methods can be used with any of the scratch comprising scheme as long as steadiness is sustained in the illustration of objects. Effective jobs that can be controlled by automatic software defined scratch memory contain static code distribution, Subject to the scheme of the target memory system , heap and dynamic data allocation, stack frame management, software-defined shared memory places (local or virtual), message passing & inter-process communication, address translation & virtual page size adjustment.

By considering the execution and target build environment a system is proposed in which the tool chain (with or without external libraries, defined as comprising of complete phases necessary for running and building an application,) is optionally allowed to obtain complete data. The layout and optional statistics about the order of dimension of the target runtime memory organization, and the desired level of scratchpad computerization (comprising the scratchpad access rules as well as capabilities implicitly) are the key parameters included. Anywhere when constraints are often mostly sensitive the latter is non-compulsory in all circumstances however can produce countless aid meant for embedded structures.

For static memory allocation scratches are mostly favorable. To further optimize the applications and the usage of runtime profiling apparatuses that would deal with practical data the approaches are previously defined that are designed for static allocation can be improved, but in the situation of difficult applications this method would need extra computational resources but not essentially assured about improve outcomes. When needed the static memory modifications can be performed manually and it is therefore recommended that for static objects whenever capacity allows allocation in a heterogeneous memory order supports scratchpads.

In the literature graph the theoretic approaches may create provably exact optimizations of stack variable distribution. Near the execution environment unique chance to increase efficiency is then to make use of supplementary data. By application-specific assumptions and/or architecture-specific features this contains techniques using which the management overhead can be additionally decreased. A lot of the run time calls need by the referenced systems comes to be needless while virtual memory management is providing as an example. Frame in a hardware-managed cache system memory inside an effective development frame may be controlled nearly identical to such a procedure, related by the procedure to avoid defaulting distribution to caches by means of the exemption that observations may obviously target scratchpad positions. By means of any appropriate arrangement of instruction-level parallelism, specific to scratchpad memory management even in architectures with hardware acceleration abilities, supposing scheduling necessities can be seen, or even by using SIMD assessment instructions can be added in architectures enhancements symmetric to the speeding up of tag processes.

At a higher level to take memory distributions equally as a data movement problem is another direction for improvement. Limited hardware managed cache coherence or for example those presenting new procedures of non-uniform memory access for more composite hybrid memory designs, the method can be stretched to maintain optimization at a system-level perspective. For data accesses to all of these dissimilar memory sections such that predictable costs can be determined a model can be parameterized, the optimization process can further be informed. From each element the expected band-width, access rules (containing latency, approvals and request routing in the case where alternating routes be present) and the arrangement of the memory order are the important system level characteristics included. To guide greedy optimization algorithms more robust cost functions or via simple estimated cost parameters these can be used to guide if the above attributes are provided. Analogous to those using commercially accessible processor generator plan flows, more detailed graphical models describing data transfers and memory access operations that would permit for further precise cost functions could also be constructed. On the target system thereof by means of profiling illustrative benchmark code and further possibility would be to develop these overheads or optimization curves. To perform design space exploration the resulting software setting comes out to be a very great tool for system engineers rather than depend only on the distinctive constant inside target explanation data, if a tool chain were capable to access these hardware explanation models.

To extent the advanced automatic heap management further improvements to assure the occurrence of bundle of data in local memory reducing overhead with methods included, when to support and called while access patterns are expected (or at least probabilistic) to maintain the prefetching of heap data. By runtime profiling or static analysis such patterns may be determined. Where deterministic access patterns are decidable (Lee et al., 2012), common to hardware prefetching and allows probable software prefetching does not experience the hypothetical overhead.

OPTIMIZATION

By the hardware on which it runs, the implementation of software memory management effectively remains primarily restricted. For competently overlapping or interleaving computation and communication the hardware embodiments suggest favorable procedures. Through double-buffering through three cache stages (Gao, 2014), on DSP platforms to efficiently mask data movement overhead among memory places the accessibility of DMA relocations was presented. Through high bandwidth on-chip networks many core designs provide similar capabilities; from data locality software on these designs significantly benefits, as power consumption and high latencies in off-chip DRAM accesses remain evaded (Mattson et al., 2008). By the balance resources (e.g. functional unit operand counts, SRAM port counts and register latency) the highest possible efficiency of memory management is basically restricted in any of these cases.

Whether incorporated in a tool chain or provided as a support libraries, it is usually required that any scratch memory management system permits optimization, that will affect by the user's capability to link code as well as debug code amongst unlike languages or operate without lacking specific source-to-source conversions or that includes the program coded in assembly language. Optimization authorizations must create the usage of every providing target account data when enabled. To the architectures of interest memory object packing are exactly fit and certain techniques in particular related to vectorization and auto-parallelization? Wherever the optimized program configuration improved plans to the target arrangement these techniques can produce code and the accessible possessions of the architecture may more efficiently use.

According to the polyhedral model (Grosser et al., 2012) one specific method of concern is that of optimization applied. Inside nested loop organizations where a bulk of execution logic be present in scientific and high performance computing uses these methods are particularly useful. Polyhedral optimization that may be measured as a commonly appropriate method however sufficient of these restrictions can be achieved by bearing random conventional rules previously it can be properly characterized in this model there are so many limitations that need to be seen by a code section.

As activities linked using loop caches or further dynamic logic might be matched by backbone through program, in this approach scratchpad memory is consider to be the most common representation of local memory to design. Our concept of a memory item is clear as one through which a superset of the above-mentioned distribution approaches stay related. Spreading further than current methodologies that attempt to simply maximize metrics of data locality throughout automatic distribution on scratch-pad schemes by suggesting relating analogous methods to the hardware conscious ne-grained management of memory objects in addition to the use of present techniques valid to converts on loop constructions.

Including loop tiling Automatic parallelization determinations have produced methods. As bandwidths and memory sizes may be trivially resolved after measured in scratchpad design is considerably reduced in complication these are the basic premise of tiling. About an execution environment if suppositions around worst-case access potential can be prepared, some classes of non-uniform, tiling alterations following all uniform, and synchronization (reliant on the memory model in consideration or pipelining) data requirement come to be somewhat available. Further if Static control flow likewise come to be more simply decidable (this concluding fact is mainly dynamic by collective core plus execution part calculations), if hardware interrupts can be supposed to happen at suitably low frequency supposed to be unimportant or are not implemented. Data dependence becomes quite accessible as these may be deprived of loss of generalization equally represented; within an execution component separately further vectorization or SIMDization is not considered.

On an iterative succession of passes the suggested technique depends. The compulsory illustrations comprising data reliance graph, particular task intermediary arrangement or an

Abstract Syntax Tree (AST), control flow graph (allowing to statically-resolved call calculations using weights given) and some related specified lower-level representations are first produced. According to the existing particulars about accessibility of resources and the execution setting then the application can initiate to be covered. At higher levels of abstraction these tiling's begin where become progressively lower level, highest communication latencies, when in-order issue is known and available.

Though effective for resolving the specific systems, the ILP designs of the above scheme may be invoked. For producing ideal outcomes a range of estimation methods may be replaced, in the cases where under time constraints or required assumptions may not be made.

CONCLUSION

For the data access requirements and by utilizing a C application as a base in a principally hardware agnostic way, a methodology for program management of scratch memory has been defined. Using new proposed approach based on previous works, a number of known techniques are combined but the elementary modules are applied in place of tool chain extensions. Totally novel techniques to optimization as well as important opportunities for upgrading of every task are identified. To develop the productivity of data movement in hybrid memory systems scratchpad memory holding configurations or its modules may possibly be used by applications; this memory management scheme should suggest significantly enhanced programmability, mutually.

REFERENCES

Bai K. and Shrivastava. A. (2000) Heap data management for limited local memory (llm) multi-core processors. In Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference, pages 317-325. IEEE.

Bai. K. (2014) Compiler and Runtime for Memory Management on Software Managed Manycore Processors. PhD thesis, Arizona State University.

Banakar, R.; Steinke, S.; Lee, B.-S.; Balakrishnan, M. and Marwedel, P. (2002) Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In Proceedings of the tenth international symposium on Hardware/software codesign, pages 73-78. ACM.

Gao. Y. (2014) Automated Scratchpad Mapping and Allocation for Embedded Processors. PhD thesis, University of South Carolina - Columbia.

Grosser, T.; Groesslinger, A. and Lengauer. C. (2012) Polly performing polyhedral optimizations on a low-level intermediate representation. Parallel Processing Letters, 22(04):1250010.

Kannan, A.; Shrivastava, A.; Pabalkar, A. and Lee, J.-e. (2009) A software solution for dynamic stack management on scratch pad memory. In Proceedings of the 2009 Asia and South Paci_c Design Automation Conference, pages 612-617. IEEE Press.

Lee, J.; Kim, H. and Vuduc. R. (2012) When prefetching works, when it doesn't, and why. ACM Trans. Archit. Code Optim., 9(1):2:1-2:29.

Lu, J.; Bai, K. and Shrivastava. A. (2013) Ssdm: smart stack data management for software managed multicores (smms). In Proceedings of the 50th Annual Design Automation Conference, page 149. ACM.

Mattson, T. G.; Van der Wijngaart, R. and Frumkin. M. (2008) Programming the intel 80-core network-on-a-chip terascale processor. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, page 38. IEEE Press.

Pena A.J. and Balaji. P. (2014) Toward the efficient use of multiple explicitly managed memory subsystems. In Cluster Computing (CLUSTER), 2014 IEEE International Conference on, pages 123-131. IEEE.

Shivaraj, K. and Dharishini, P.P.P. (2015) Design and simulation analysis of time predictable computer architecture. MSRUAS-SASTech Journal, 14(1):5-8.

Verma, M.; Wehmeyer, L. and Marwedel. P. (2005) Efficient scratchpad allocation algorithms for energy constrained embedded systems. In Power-Aware Computer Systems, pages 41-56. Springer.

--0--

Manuscript Submission Date: **20 February 2020**

Revised Submission Date: **23 April 2020**

Date of Acceptance: **05 May 2020**