Um Estudo Preliminar do Uso de Memória Persistente Através da Biblioteca Mnemosyne

Rafael Pizzirani Murari[1], **Alexandro José Baldassin**[1]

[1]Programa de Pós-Graduação em Ciência da Computação –
Universidade Estadual Paulista (UNESP)
Rio Claro – SP – Brasil

{rpmurari,alex}@rc.unesp.br

Abstract. *The recent non-volatile memory technologies have caused a disruption in computational systems, making it possible to reduce the overhead related to data persistence. However, some precautions are required to ensure data consistency in occasional power outage or system failures. This paper presents a preliminary study of persistent memory usage through the sofware solution Mnemosyne, responsible for ensuring consistency through a transactional support.*

Resumo. *As recentes tecnologias de memória não-volátil tem causado uma disrupção nos sistemas computacionais, possibilitando a redução do overhead associado a persistência de dados. Entretanto são necessárias algumas precauções para manter a consistência destes em eventuais quedas de energia ou falhas de sistema. Este trabalho apresenta um estudo preliminar do uso de memória persistente através da solução em software Mnemosyne, responsável por garantir a consistência através de um suporte transacional.*

1. Introdução

As emergentes tecnologias de memória não-volátil, conhecidas como *Non-Volatile Memory* (NVM), são caracterizadas pela durabilidade, endereçamento por byte, acesso via instruções de memória (*read* e *write*) e tempo de acesso semelhante à memória DRAM (*Dynamic Random Access Memory*). Tais características possibilitam a organização da hierarquia de memória com uma memória principal híbrida, constituída pela DRAM (volátil) e pela NVM (não-volátil), conectadas diretamente no barramento de memória. Porém a simples adição da NVM no barramento de memória não possibilita a extração do máximo potencial desta, pois apenas parte do *overhead* associado ao caminho crítico dos dados, como a serialização destes e a utilização de *drivers* para comunicação através do barramento de I/O, é removido.

As camadas intermediárias presentes na pilha de software, como o sistema de arquivos e o *Kernel*, também adicionam *overhead*, e consequentemente deterioram o desempenho de aplicações com grande volume de atualizações em dados persistentes. O mapeamento de regiões de memória persistente no espaço de endereçamento dos processos possibilita eliminar essas camadas, tornando a persistência dos dados mais ágil, porém tal procedimento não garante a consistência destes em eventuais falhas de sistema ou quedas de energia.

As inconsistências são originadas pelo fato de a memória cache, presente nos processadores contemporâneos, adotar a política de *write-back*, caracterizada por atualizar

os dados somente na cache, copiando estes para a memória somente quando o bloco for substituído na cache. Podemos destacar duas adversidades, sendo a primeira delas a distribuição dos dados entre a cache e a memória, onde uma eventual queda de energia acarretaria a perda de informações presentes na cache, e consequentemente deixando a memória em um estado inconsistente, como por exemplo a perda de parte de uma tabela hash persistente. A segunda é a desordenação das escritas em memória, onde a perda de escritas precedentes às atualizações feitas em memória pode causar corrupção dos dados ou vazamento de memória em ocasionais falhas.

Neste contexto, este trabalho apresenta a solução em software Mnemosyne [Volos et al. 2011], responsável por expor a memória persistente ao programador através do suporte transacional oferecido pela mesma. O objetivo final do trabalho de mestrado é acelerar a execução do Mnemosyne através de Memória Transacional em Hardware (HTM). Este texto, no entanto, foca em uma etapa intermediária onde o uso do Mnemosyne em sistemas contemporânos, que utilizam o *GNU Compiler Collection* (GCC), é investigado. O artigo é dividido da seguinte forma: a Seção 2 descreve a biblioteca Mnemosyne, a Seção 3 apresenta as modificações necessárias para se utilizar a versão portada para o GCC. Por fim, a Seção 4 apresenta as conclusões obtidas.

2. Mnemosyne

O Mnemosyne é um sistema leve, composto por duas bibliotecas no nível de usuário (*libmnemosyne* e *libmtm*), responsável por expor a memória persistente ao programador através de uma interface simples, abstraindo a complexidade de gerenciamento e oferecendo suporte transacional, capaz de prover atualizações atômicas e duráveis de múltiplos dados persistentes.

A Memória Transacional [Harris et al. 2010] é uma abordagem para o desenvolvimento de aplicações paralelas baseada nos mesmos conceitos de transações em banco de dados, guiadas pelas quatro propriedades denominadas ACID: atomicidade, consistência, isolamento e durabilidade. As transações devem ser realizadas de maneira atômica, ou seja, todas as instruções presentes no bloco atômico devem ser efetivadas no sistema ou, em caso contrário, as alterações são desfeitas e a transação abortada. Os passos intermediários das transações não são visíveis às demais transações, garantindo assim isolamento entre estas. A efetivação da transação deve garantir a transição de um estado previamente consistente para um novo estado consistente de memória. Por fim, os dados efetivados pelas transações são duráveis, ou seja, são recuperáveis em eventuais quedas de energia.

A utilização da Memória Transacional possibilita a solução do primeiro problema de inconsistência de dados, apresentado na Seção 1, porém ainda não é capaz de resolver o segundo. Para tal, o Mnemosyne disponibiliza uma barreira de persistência, composta pelas instruções *mfence* e *clflush*, apta a manter a ordem das escritas em memória realizadas pelas aplicações. A instrução *mfence* previne que escritas subsequentes a sua chamada sejam completadas antes das precedentes, enquanto a instrução *clflush* é responsável por invalidar a linha de cache contendo um determinado endereço em todos os níveis da cache, forçando a escrita dos dados na memória antes de completar a invalidação.

A utilização de Memória Transacional e da barreira de persistência não é exclusiva ao Mnemosyne, outras soluções em softwares também disponibilizam tais propriedades,

como o NV-Heaps [Coburn et al. 2011], o NVL-C [Denny et al. 2016] e o NVML [Pme], porém o Mnemosyne apresenta uma sintaxe mais simplificada e menos propensa a erros, como pode ser observado no Código 1, onde é apresentado um exemplo de transação constituída da alocação de um novo elemento e sua respectiva inserção em uma tabela hash persistente.

Código 1. Exemplo de transação durável no Mnemosyne.

```
MNEMOSYNE_PERSISTENT hash_t hashtable;

update_hash(int key, int value) {
  atomic {
    bucket_t *bucket = pmalloc(sizeof(bucket_t));
    bucket->key = key;
    bucket->value = value;
    insert(hashtable, bucket);
  }
}
```

3. Modificações

O Mnemosyne foi originalmente desenvolvido utilizando-se o compilador *Intel C++ Compiler* (ICC) com suporte a Memória Transacional em Software (STM), porém este encontra-se descontinuado. Neste contexto, esta ferramenta foi portada para o GCC, entretanto algumas alterações são necessárias no desenvolvimento das aplicações utilizando-se esta nova versão, devido essencialmente a divergências no modo em que as variáveis persistentes são referenciadas pelos compiladores.

A fase de inicialização da biblioteca Mnemosyne executa uma rotina de verificação de eventuais execuções anteriores de aplicações, mapeando os dados previamente alocados para o espaço de endereçamento do processo. Após o mapeamento, é necessário associar as respectivas variáveis persistentes, presentes no arquivo objeto, com os endereços mapeados.

Conforme pode ser observado no Código 1, a declaração de variáveis persistentes é realizada através da palavra reservada `MNEMOSYNE_PERSISTENT`. Todas as variáveis declaradas deste modo são colocadas na seção `.persistent` do arquivo objeto, no formato *Executable and Linking Format* (ELF), gerado pelo processo de compilação. O ICC referencia as variáveis definidas nesta seção através da *Global Offset Table* (GOT), enquanto o GCC utiliza o endereço associado a esta região.

A GOT é uma tabela, localizada na seção de dados, utilizada para a resolução do referenciamento de endereços de variáveis pelas instruções presentes em uma biblioteca dinâmica, devido a esta poder ser mapeada em diferentes localizações no espaço de endereçamento dos processos. As instruções apontam para entradas na GOT e, após determinado o endereço absoluto da variáveis, basta atualizar as posições respectiva a estas na tabela, evitando percorrer todo o código e atualizar os endereços nas instruções, processo conhecido como relocação.

Neste contexto, a solução abordada é a realização de declarações de variáveis persistentes em bibliotecas dinâmicas, sendo estas disponibilizadas às aplicações através de uma *Application Programming Interface* (API), evitando o *overhead* do processo de relocação dos endereços. Esta solução se mostrou efetiva, porém acrescentou um encargo maior ao programador, sendo necessário reestruturar o código através da inserção

de bibliotecas dinâmicas.

4. Conclusão

Neste trabalho foram apresentadas as características das emergentes tecnologias de memória não-volátil e seu respectivo impacto na arquitetura do sistema e na pilha de software. A garantia de consistência dos dados é o maior desafio a ser confrontado, visto que eventuais falhas do sistema ou quedas de energia ocasionam a perda de informações, necessitando uma estratégia para manter a corretude destas.

Com esta finalidade, foi apresentada a solução em software Mnemosyne, responsável por prover suporte transacional e gerenciamento de memória, capaz de garantir a consistência e durabilidade do conjunto de dados das aplicações, apesar do encargo adicionado na versão portada ao GCC.

A próxima etapa deste trabalho é averiguar a aceleração do Mnemosyne através de HTM, presente em processadores recentemente disponibilizados pela Intel e IBM.

Agradecimentos

Os autores agradecem ao suporte fornecido pela Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Referências

pmem.io : Persistent memory programming blog. `http://pmem.io/`. Acesso em: 16 de fev. 2017.

Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., and Swanson, S. (2011). Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 105–118, New York, NY, USA. ACM.

Denny, J. E., Lee, S., and Vetter, J. S. (2016). Nvl-c: Static analysis techniques for efficient, correct programming of non-volatile main memory systems. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, pages 125–136, New York, NY, USA. ACM.

Harris, T., Larus, J., and Rajwar, R. (2010). *Transactional Memory*. Morgan and Claypool Publishers, 2nd edition.

Volos, H., Tack, A. J., and Swift, M. M. (2011). Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 91–104, New York, NY, USA. ACM.