Artigo Visão: Processamento de Banco de Dados em Memória

Abstract. *Neste artigo, apresentamos nossa visão de como os Sistemas Gerenciadores de Bancos de Dados (SGBD) podem integrar Processamento-em-Memória (PIM) em processamento de consultas. PIM promete mitigar os problemas clássicos de memory-wall e energy-wall presentes nas arquiteturas de computadores que são amplificados pela movimentação de dados na hierarquia de memória por aplicações de análise de dados. Compartilhamos com a comunidade uma análise empírica dos prós e contras do uso de PIM em três principais operadores da algebra relacional: seleção, projeção e junção. Com base nos resultados obtidos começamos a develvolver um escalonador de consultas PIM-consciente que apresenta resultados promissores chegando a reduzir em 3× o tempo de execução de consultas e o consumo de energia em pelo menos 25%. Concluímos nossa visão com uma discussão sobre os desafios e oportunidades para impulsionar pesquisas na concepção de um SGBD com PIM.*

1. Introdução

PIM foi originalmente concebido no final da década de 60 [Kautz 1969] e consiste em adicionar unidades de processamento dentro do dispositivo de memória para uso eficiente da largura de banda interna. Ao longo dos anos, a ideia de PIM foi revisitada de acordo com o progresso das tecnologias de memória. Por exemplo, componentes de processamento foram adicionados em discos magnéticos para executar certos algoritmos de processamento de consultas [DeWitt and Hawthorn 1981, Keeton et al. 1998]. O mesmo ocorreu às tecnologias de memória RAM [Patterson and et al. 1997] que tentaram adicionar unidades lógicas dentro da DRAM para realizar computações específicas. Infelizmente, produtos comerciais não adotaram essas abordagens devido às limitações tecnológicas de hardware e porque o crescimento contínuo de desempenho de CPU obedecia a lei de Moore e o escalonamento de Dennard. Soluções em discos flash também tentaram embutir unidades funcionais [Do et al. 2013]. Porém, essas soluções não tiveram êxito devido à falta de uma interface de programação genérica e o custoso tratamento de erros de hardware.

Sistemas centrados em dados movem grandes quantidades de dados pela hierarquia de memória até a CPU. Por isso, novas arquiteturas PIM emergem devido à introdução das vias de silício em memória (*Through-Silicon Vias* (TSVs)) que tornaram as memórias 3D viáveis: 3D consiste na integração de chips de DRAM e células lógicas no mesmo dispositivo. Na prática, as memórias 3D invertem o modelo de processamento tradicional, i.e., elas movem a computação até os dados. Atualmente, as GPUs comerciais embutem memórias 3D nas tecnologias: Cubo Híbrido de Memória (HMC) [Jeddeloh and Keeth 2012] e Memória de Alta Largura de Banda (HBM) [Kim and Kim 2014].

Portanto, PIM surge como uma abordagem atrativa para reduzir a movimentação de dados em sistemas centrados em dados. Logo, em nossa visão, já é tempo de discutir como SGBDs podem odotar PIM no processamento de consultas. Os maiores benefícios

a serem explorados são a redução drástica no consumo de energia e a alta largura de banda interna dos novos dispositivos, pois estes fornecem um alto nível de acesso paralelo aos dados aliado ao processamento em memória. Atualmente, PIM tem sido usado de forma isolada como acelerador de operadores, tais como: seleção [Tome et al. 2018b] e junção [Mirzadeh et al. 2015]. Porém, notamos que o comportamento de cada operador depende de características do conjunto de dados e das configurações do sistema de caches. Neste artigo discutimos os seguintes aspectos para o desenvolvimento de um SGBD-PIM:

I. Idealizamos um processador de consulta PIM-consciente para usufruir do paralelismo intra-consulta entre CPU e PIM para responder a pergunta: *Quando é melhor processar os dados no dispositivo de memória ao invés de movê-los pela hierarquia de memória até a CPU?*

II. Introduzimos um escalonador PIM-consciente que visa responder a pergunta de pesquisa: *Como um SGBD pode coordenar a execução intra-consulta entre CPU e PIM para melhor explorar o potencial de cada dispositivo de processamento?*

III. Divulgamos resultados preliminares sobre o impacto de PIM no processamento tradicional de consulta e discutimos a economia de energia junto com a redução drástica no tempo de execução além de uma ordem de magnitude para certos operadores.

IV. Compartilhamos uma lista de desafios e oportunidades para o desenvolvimento e integração de um SGBD-PIM.

2. Fundamentos de Arquiteturas PIM

As memórias 3D emergentes integram quatro ou oito chips de DRAM empilhados e interconectados via TSV até uma camada lógica na base. A pilha de chips de DRAM é organizada de forma vertical, formando partições chamadas "*vaults*" (Figura 1). As memórias 3D atuais possuem 32 vaults interdependentes para acessar e processar dados em paralelo alavacando a largura de banda interna em até 320 GB/s, e suportam instruções aritméticas, lógicas e binárias com operandos de até 16-bytes.

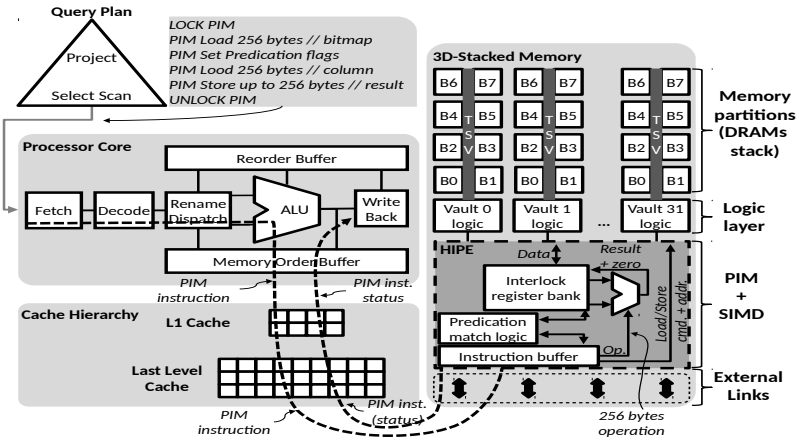

Figura 1. A query datapath movement in a traditional von Neumann architecture plus a modern 3D-stacked memory with PIM and SIMD support.

O trabalho recente de [Tome et al. 2018b] apresenta uma extensão da arquitetura desses dispositivos para operar com instruções SIMD (*Single Instruction, Multiple Data*)

em vetores de 256-bytes para processamento de consultas, pois cada *vault* possui um buffer de 256-bytes para acessar os bancos de memória correspondentes a sua partição. A Figura 1, lado direito, ilustra essa arquitetura PIM com suporte SIMD e, no lado esquerdo, a arquitetura von Neumann tradicional, i.e., o processador com a hierarquia de caches separados. No topo da Figura 1, o processador dispara instruções PIM diretamente ao dispositivo PIM contornando a hierarquia de caches. No final, o dispositivo PIM retorna somente o status da instrução à CPU para continuar a execução do pipeline. Esse mecanismo é a principal vantagem sobre atuais SGBDs (e.g. Netezza and Exasol) que também filtram dados em hardware antes de passar pela CPU. Porém, eles precisam condensar os dados filtrados em páginas para evitar a poluição do *bufferpool* que é custoso e propenso a erros. Por outro lado, os protocolos atuais de memória lidam com todas as idiossincrasias de instruções PIM, tais como: coerência de cache, correção de erros de códigos de memória (ECC) e acesso direto à memória (DMA). O fluxo de execução funciona no nível de instrução como o processamento tradicional da CPU (e.g., AVX/SSE): os programadores adicionam instruções PIM intrínsecas, como as *Intel Intrinsics*, e o compilador marca elas como instruções de memória especiais.

3. Processamento de Consulta com PIM

Para avançarmos no desenvolvimento do processamento de consulta PIM-consciente precisamos entender o impacto do uso do PIM na execução de consulta tradicional. Por isso, analisamos tal impacto em uma carga de trabalho analítica ao comutar o processamento entre CPU e PIM em termos do tempo de execução e consumo de energia dos operadores de projeção, seleção e junção. Realizamos nossa análise através do simulador SiNUCA devido a limitações de hardware e usamos os mesmos parâmetros aplicados por trabalhos relacionados [Tome et al. 2018b] e estimamos o consumo de energia com os mesmos parâmetros de DRAM do estado da arte [Jeddeloh and Keeth 2012]. Além de tudo, o SiNUCA tem sido extensivamente adotado em artigos científicos em arquitetura de computadores [Alves et al. 2016] e banco de dados [Tome et al. 2018a, Kepe 2018].

Avaliamos os operadores através do *benchmark* TPC-H de 1GB porque os conjuntos de entrada cabem em cache que é o melhor cenário para processamento em CPU. As implementações dos operadores usam operações SIMD de 64-bytes (CPU/AVX-512) e 256-bytes (PIM). Os resultados convergem para explorarmos nossa visão de SGBD-PIM.

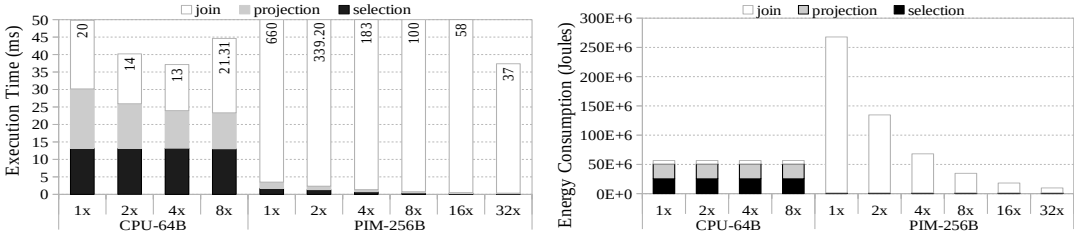

Figura 2. Tempo de execução e consumo de energia dos operadores da consulta 03 do TPC-H: seleção, projeção e junção em CPU-64B e PIM-256B com profundidades de *loop* desenrolados entre 1× até 32×.

Projeção: A Figura 2 mostra que a projeção em PIM reduz o tempo de execução em mais de uma ordem de magnitude, aproximadamente 60×. Também observamos uma redução de consumo de energia em 36×. **Seleção**: A seleção com PIM reduz o consumo de energia em 98% e o tempo de execução em 76× comparado a execução em CPU.

Os resultados são devido ao comportanemto em *streaming* dos operadores com PIM que causa baixo reuso de dados e pouca movimentação destes fora da memória principal. No processamento em CPU, as colunas são transferidas desde a memória principal passando por toda hierarquia de caches até a CPU que compara porções das colunas com os predicados de seleção e projeção. Porém, todas as porções de dados transferidas não são reutilizadas durante a execução, inibindo o mecanismo de *caching*. Em contrapartida nos operadores em PIM, a CPU envia apenas as instruções para o dispositivo PIM. Os operadores acessam e avaliam os dados com até 32 instruções PIM+SIMD de 256-bytes.

Junção: Em nossos experimentos, implementamos três algoritmos de junção, porém os algoritmos de Hash Join e Sort-Merge Join geram acessos aleatórios à memória que inibem o acesso paralelo aos dados e a alta largura de banda dos dispositivos PIM. Por isso, para fornecer uma percepção sobre os problemas de junção em PIM, nós analisamos a execução do Nested Loop Join (NLJ). O NLJ-PIM desenrola o *loop* interno até $32\times$ para explorar os acesso paralelo aos dados. Porém, cada iteração desse *loop* causa instruções compulsórias de leitura e escrita na memória, pois a coluna de junção interna (*inner column* em cache) é reacessada em toda iteração do *loop*. Por isso, o processamento em CPU é melhor que a execução em PIM, chegando a ser $2.8\times$ mais rápido, conforme mostra o gráfico da Figura 2. Mesmo o consumo de energia, o maior benefício de usar PIM, é 70% mais custoso em PIM. Entretanto, o NLJ-PIM torna-se viável quando a coluna interna de junção não cabe em cache o que inibe o reuso de dados. Neste ponto, o PIM obtêm uma melhora de $1.38\times$ comparado ao processamento em CPU.

4. Processamento de Consulta PIM-Consciente

Em nosso estudo preliminar sobre o impacto de PIM no processamento consulta, nós observamos que nem todo operador de banco de dados se beneficia do PIM. Portanto, a decisão sobre a arquitetura de processamento de um operador de consulta não é trivial. De fato, observamos que os operadores de consulta com alta reutilização de dados se beneficiam do mecanismo de *caching* e, assim, o processamento em CPU torna-se atraente, como no caso do NLJ. Por outro lado, os operadores com comportamento *streaming* (e.g., projeções e seleções) são mais adequados para o PIM. Outros operadores, como a agregação, podem ter um reuso de dados baixo ou médio e um comportamento de *streaming* parcial, o que torna pouco evidente qual é o melhor dispositivo de processamento.

Nossa investigação concentra-se em como intercalar a execução de consultas entre a CPU e o PIM. Até onde sabemos, esse é o primeiro esforço nessa direção. Iniciamos nosso estudo com um escalonamento estático que usa um modelo de classificação baseado no perfil de operadores para decidir em qual arquitetura processar um dado operador. Diferente de trabalhos relacionados em escalonamento para GPU, esta estratégia não requer uma fase de calibração [Breß et al. 2012] nem heurística [Karnagel et al. 2014]. A Figura 3 apresenta três planos de execução de consulta a partir do plano ótimo: dois planos hardwares-específicos, i.e., CPU e PIM, e um plano híbrido baseado no escalonamento estático de perfil. No plano híbrido, os operadores de seleção e projeção são processados em PIM, enquanto que a junção é executada em CPU. Com o plano híbrido a execução de consulta tem uma melhora em quase $3\times$ (Figura 3d), pois explora as vantagens de ambas as arquiteturas: operadores com alto reuso de dados são executados na CPU e operadores *streaming* são processados em PIM. Em termos de consumo de energia, o plano híbrido chega a reduzir em pelo menos 25%, Figura 3e.

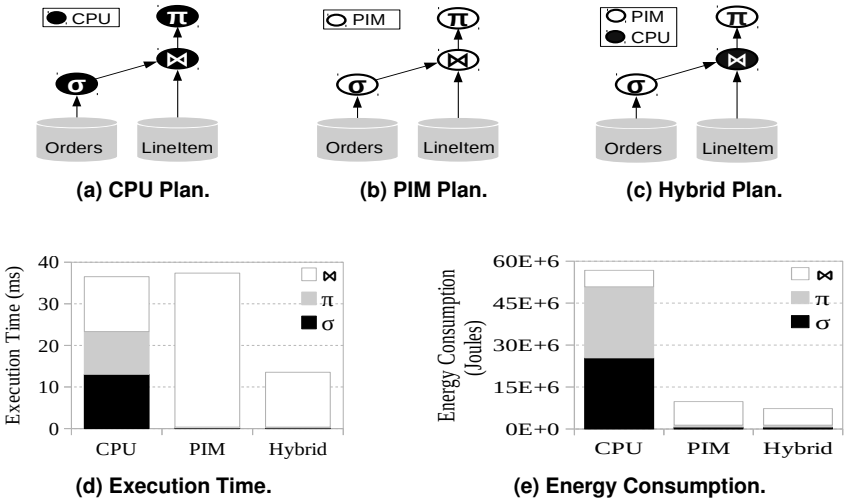

Figura 3. Planos de consulta: planos isolados CPU (a) e PIM (b), plano híbrido (c). Respectivos tempo de execução (d) e consumo de energia (e).

5. Desafios & Oportunidades

Co-processamento Simultâneo: Em ambientes de processamento heterogêneos, o otimizador precisa identificar oportunidades de co-processamento para evitar dispositivos ociosos ou consumo de energia ineficiente (e.g., o problema do muro de energia [Wang and Skadron 2013]). Além disso, a detecção de concorrência à memória é difícil de gerenciar no processamento simultâneo entre CPU e PIM.

Otimização do Plano de Consulta: O espaço de busca na otimização de planos de consulta já é um problema comum ao otimizador de consulta. A adição de planos híbridos aumenta ainda mais a complexidade para gerar planos de consuldas cadidatos. Desta forma, um otimizador de consulta PIM-consciente deve considerar características específicas de hardware para escolher o dispositivo de processamento apropriado.

Transações: Intrisicamente, as instruções PIM aritiméticas e lógicas são atômicas [HMC-Consortium 2015]. O que proporciona oportunidades de pesquisa para transações em PIM e *Hybrid Transactional/Analytical Processing (HTAP)*. A ISA atual dos dispositivos PIM suportam instruções do tipo comparação-e-troca para teste de dados. Por conseguinte, instruções de atualização PIM podem ser sincronizadas em memória sem necessitar a checar as caches ou uso extra da banda de memória. A oportunidade é reduzir o *overhead* de bloqueio de dados e de estruras (*locking* e *latching*) que correspondem a 30% das intruções OLTP [Harizopoulos et al. 2008].

Adoção do SGBD: Em nossa visão SGBDs devem invocar intruções PIM no código dos operadores, similarmente as abordagem SSE e AVX. Nós também consideramos otimização de código para incluir funções intrinsics na ISA dos dispositivos PIM.

6. Conclusão & Trabalho Futuro

Neste artigo apresentamos nossa visão de um SGBD que explora a largura de banda de memória sem precedentes dos novos dispositivos PIM aliada ao processamento em memória. Nós discutimos resultados promissores ao intercalar a execução de processamento intra-consulta entre PIM e CPU. Com nosso escalonador estático, planos de

consulta híbridos melhoraram o desempenho cerca de 3× e reduziram o consumo de energia em cerca de 25%. Apesar desses resultados promissores, sabemos que existem limitações a serem superadas. Por isso, nosso trabalho em andamento consiste em um escalonador dinâmico para criar e atualizar perfis e reescalonar operadores dinamicamente. Também consideramos investigar o algoritmo Hash Join, Sort-Merge Join e agregações. Por último, compartilhamos com a comunidade uma lista de desafios e oportunidades abertas pela nossa visão para o *co-design* do SGBD-PIM.

Referências

Alves, M. A. Z., Diener, M., Santos, P. C., and Carro, L. (2016). Large vector extensions inside the HMC. In *DATE*.

Breß, S., Mohammad, S., and Schallehn, E. (2012). Self-tuning distribution of db-operations on hybrid CPU/GPU platforms. In *24th Grundlagen von Datenbanken*.

DeWitt, D. J. and Hawthorn, P. B. (1981). A performance evaluation of data base machine architectures (invited paper). In *7th VLDB*.

Do, J., Kee, Y., Patel, J. M., Park, C., Park, K., and DeWitt, D. J. (2013). Query processing on smart ssds: opportunities and challenges. In *SIGMOD*.

Harizopoulos, S., Abadi, D. J., Madden, S., and Stonebraker, M. (2008). OLTP through the looking glass, and what we found there. In *SIGMOD*.

HMC-Consortium (2015). *Hybrid Memory Cube Specification 2.1*. HMC-30G-VSR PHY.

Jeddeloh, J. and Keeth, B. (2012). Hybrid memory cube new DRAM architecture increases density and performance. In *VLSIT*.

Karnagel, T., Habich, D., Schlegel, B., and Lehner, W. (2014). Heterogeneity-aware operator placement in column-store DBMS. *Datenbank-Spektrum*.

Kautz, W. H. (1969). Cellular logic-in-memory arrays. *IEEE Trans. Computers*.

Keeton, K., Patterson, D. A., and Hellerstein, J. M. (1998). A case for intelligent disks (idisks). *SIGMOD Record*.

Kepe, T. R. (2018). Dynamic database operator scheduling for processing-in-memory. In *PhD@VLDB*.

Kim, J. and Kim, Y. (2014). HBM: memory solution for bandwidth-hungry processors. In *26th HCS*.

Mirzadeh, N., Kocberber, O., Falsafi, B., and Grot, B. (2015). Sort vs. hash join revisited for near-memory execution. In *ASBD@ISCA*.

Patterson, D. A. and et al. (1997). A case for intelligent RAM. *IEEE Micro*.

Tome, D. G., Kepe, T. R., Alves, M. A. Z., and de Almeida, E. C. (2018a). Near-data filters: Taking another brick from the memory wall. In *ADMS@VLDB*.

Tome, D. G., Santos, P. C., Carro, L., de Almeida, E. C., and Alves, M. A. Z. (2018b). HIPE: HMC instruction predication extension applied on database processing. In *DATE*.

Wang, L. and Skadron, K. (2013). Implications of the power wall: Dim cores and reconfigurable logic. *IEEE Micro*.