統合型 GPU を用いた hugepage zerofilling の高速化

巴統哉 [1]　河野健二 [1]

概要：統合型 GPU（integrated GPU) は現在企業や家庭などで一般的に使用されているにも関わらず，ほとんどのマシンにおいて，使用率が低くなっている．これは，一般的な環境では，デスクトップやブラウジングなど, 最低限の描画処理しか行われず, 高負荷の描画処理や GPGPU の演算などは不要なためである．本研究では, Linux kernel 内処理を integrated GPU へオフロードする処理を作成し，Linux kernel 内処理である hugepage の zerofiling 処理を GPU にて実行することで，統合型 GPU を有効活用し，CPU の負荷を減らす.

キーワード：Heterogeneous System, GPGPU, Hugepage

1. はじめに

統合型 GPU（integrated GPU) は企業や家庭などで一般的に使用されている．しかしながら, ほとんどのマシンにおいて，使用率が低くなっている．これは，一般的な環境では，デスクトップやブラウジングなど，最低限の描画処理しか行われず，高負荷の描画処理や GPGPU の演算などは行われないためである．

Integrated GPU を有効活用する既存研究として，APUNet[1] が挙げられる．APUNet ではネットワークのパケット処理を integrated GPU にて並列演算することによって，CPU の負荷を下げつつ，高速化した．

また，Linux Kernel 内には，非同期処理で処理すべきなのにもかかわらず, 直列演算で処理をしているために，レイテンシが大きく増え，全体のシステムパフォーマンスに影響を与えてしまっているタスクが多くある．一つの例として，Hugepage のゼロ埋め処理が挙げられる．Hugepage のゼロ埋めは, 465us かかっており，ページフォルトレイテンシの大半を占めていた．Ingens[2] では，このゼロ埋め処理を別のカーネルスレッドにオフロードし，非同期で処理をすることで, ページフォールトレイテンシを 3.5us まで削り, 全体のパフォーマンスを向上させた．

GPU を Linux kernel 内で利用する既存研究では，Catalyst[3] が挙げられる．Catalyst では，Kernel Samepage Merging (KSM) を分離型 GPU(discrete GPU) にオフロードすることによって，メモリを多く使用するワークロードにおいて，KSM の実行時に，CPU の負荷を 0.7 倍ほどに下げた．

Catalyst では，Discrete GPU を用いて高速化していたが，Discrete GPU では PCI-e 転送が発生してしまい，レイテンシが増えてしまう．レイテンシが重要となる Linux Kernel 内の処理において，GPU への転送は，解決すべき大きな問題であると Catalyst において示されていた．また，Linux Kernel 内では，Discrete GPU が必要なほどの高負荷の演算は多くない.

それらを考慮し，本研究では， integrated GPU を用いて Linux Kernel 内処理をオフロードし，CPU-GPU 間のメモリ転送が発生させず，パフォーマンスを向上させることを目標とした.

Linux Kernel 内処理の中で, 非同期で処理すべきタスクとして， Hugepage のゼロ埋め処理が挙げられる．Ingens では，カーネルスレッドを用いて非同期にゼロ埋めを実行していたが，結局はそのカーネルスレッドでゼロ埋めが行われるので，CPU の合計の負荷は変わらない．本研究では Ingens でのカーネルスレッドでの処理に値する部分を GPU Kernel にて行うことにより, 非同期実行した上で CPU の負荷を減らすことができる.

本研究では, Linux Kernel 内処理を integrated GPU へオフロードするフレームワークを実装し，定量的に評価する．まず，実装したフレームワーク単体のオーバーヘッドを計測し，元のゼロ埋め関数と比較した．その結果, 2M Byte のゼロ埋めの際，元のゼロ埋め関数 clear_page 関数が 0.35 M cycle かかっていたのに対し，本実装では，27 cycle まで抑えることができた．本実装を適用することにより，全体の

1 慶應義塾大学
Keio University

CPU 負荷が下がり，同じコアで動く他のアプリケーションを高速化することできる．本稿では，jhash, AES 関数を同じコアで動かし，それらが本実装により，どれほど高速化するかを評価した．

2. 実装

Hugepage のゼロ埋めを integrated GPU へオフロードするために，図 1 のようなフレームワークを考えた．GPU のカーネルをラウンチするために，Helper damon を busy wait させておき，mm/hugetlb.c からのリクエスト待つ．Helper damon では，GPU の仮想アドレスを確保し，hugepage アドレスの物理アドレスを remap_pfn_range によりその仮想アドレスへ remap する．そして，ページフォルトの際，hugepage のゼロ埋めを行なっている mm/hugetlb.c から，GPU カーネルのラウンチリクエストを行い，Helper damon がそれを受け取り，Opencl カーネルを実行する．Opencl カーネルは，hugetlb.c の clear_page 関数の代わりに，ゼロ埋めを並列で行う．

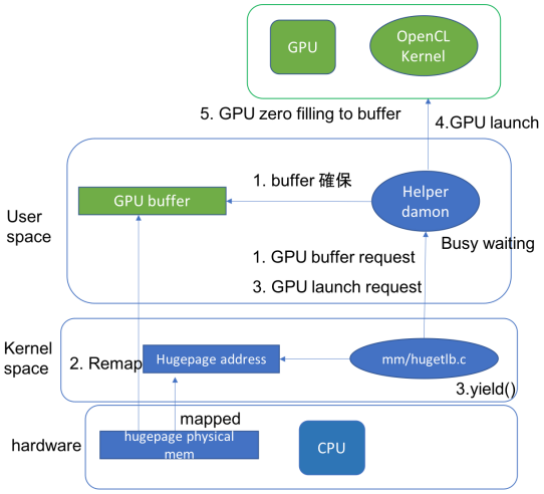

図 1 Hugepage zerofilling の GPU へのオフロード

3. 実験

まず，マイクロベンチマークとして，今回作成したフレームワークの中にタイムスタンプカウンタを置き，オーバーヘッドとなりうる主な処理の時間を測定した．今回は，GPU kernel のラウンチにかかる時間を計測した．(GPU kernel 自体の時間が非同期であるため，計算しない)

次に，このフレームワークを適用したことにより，同じコアで動いている他の処理が高速化すると考えた．そのため，以下の実験を行なった．ある値をハッシュ化する jhash 関数と暗号化する AES 関数をフレームワークと同じコアで動かし，CPU に負荷をかける．本研究のフレームワークによって CPU の負荷が減った場合，それらの関数にかかる時間が短縮すると考え，それらにかかる時間を測定した．

4. 実験結果

2Mbyte のデータをゼロ埋めする際のレイテンシを比較したところ，元の clear_page 関数では 0.35M cycle，本実装では 27 cycle だった．

これにより，約 0.34M cycle 分の CPU 処理を同じコアの他の処理に使用できる．

Jhash 関数と AES 関数において，元の Linux Kernel 上で計測したレイテンシと，本実装を適用した際のレイテンシは表 1 のようになった．また，標準偏差を追記した．

表 1 jhash, AES 関数へ本実装が与える影響

	Original [/Mcycle]	Proposal [/Mcycle]	Original (std)	Proposal (std)
Jhash 100 loop	3.00	2.91	0.161	0.00883
AES 1 loop	0.410	0.398	0.0827	0.0654
AES 10 loop	2.55	2.47	0.227	0.0116

結果から，理論値ほどは他の処理は高速化していないが，大きく性能が向上していることがわかる．理論値ほど高速化しなかった理由として，Helper damon, mm/hugetlb.c 間でコンテクストスイッチが複数回発生してしまうことが挙げられる．割込みを利用することで，将来的には簡単に解決するだろうと考えられる．

5. まとめ

使用率の低い統合型 GPU (integrated GPU) を有効活用し，Hugepage のゼロ埋めのレイテンシを減らすことで，OS 全体のパフォーマンスを向上させ，他のアプリケーションを高速化することができた．

6. 謝辞

本研究は，JST, CREST, JPMJCR19F3 の支援を受けたものである．

参考文献

[1] Y. Go, M. A. Jamshed, Y. Moon, C. Hwang, and K. Park. APUNet: Revitalizing GPU as Packet Processing Accelerator. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI '17, 2017.

[2] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. 2016. Coordinated and Efficient Huge Page Management with Ingens. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16). USENIX Association, Berkeley, CA, USA, 705–721. http://dl.acm.org/citation.cfm?id=3026877.3026931

[3] G. Anshuj, M. Debadatta, and K. Purushottam. 2017. Catalyst: GPU-assisted rapid memory deduplication in virtualization environments. In VEE. 44–59.

[4] Chia-Heng Tu and Te-Sheng Lin. 2019. Augmenting Operationg Systems with OpenCL Accelerators. In ACM Transactions on Design Automation of Electronic Systems (TODAES), New York, NY, USA, ACM, pp. 1084-4309(online), doi:10.1145/3315569