Meten en inperken van het koudestart-probleem by het simuleren van GPUs

Jonas Sys[1], Mahmood Naderan-Tahan[2], Seyyed Hossein SeyyedAghaei Rezaei[2], Lieven Eeckhout[2]

Abstract

Elk jaar worden er meer en meer AI- en machine learning-artikelen gepubliceerd [1]. Veel van deze machine learning-algoritmen vertrouwen op GPU's om hun modellen te trainen. Bovendien profiteren veel andere workloads, variërend van moleculaire simulatie tot graaf-algoritmen, enorm van de parallelle verwerkingskracht van GPU's. Deze toenemende vraag naar parallelle verwerking heeft geleid tot een grotere nood aan innovatie, waarbij GPU's voortdurend worden verbeterd. Om te controleren dat deze veranderingen effectief verbeteringen zijn, worden simulators gebruikt [2]. Moderne technieken voor simulatie vertrouwen op *sampling* [3, 4]. Deze *sampling*-technieken gaan er vaak van uit dat de hardware bij het begin perfect opgewarmd is, maar dat is niet altijd het geval. In dit artikel zullen we de impact van het koudestart-probleem op GPU-*workloads* (zowel in hardware als in de simulator) meten en enkele verbeteringen voorstellen.

1. Introductie

Sinds hun uitvinding in 1968 door de *Evans and Sutherland Computer Corporation* zijn GPU's al veel veranderd [5]. De *fixed-function pipelines* van vroeger zijn vervangen door de programmeerbare shaders van vandaag. Machines die vroeger zo groot waren als een kamer, zijn nu kleine chips die op CPU's kunnen worden ingebed.

In 2007 werd het CUDA-platform geïntroduceerd [6, 7], dat meer algemene berekeningen op GPU's mogelijk maakte. Dit was een keerpunt voor de GPU-industrie, omdat het gebruik van GPU's in een breed scala van toepassingen mogelijk maakte. Vandaag de dag worden GPU's gebruikt in vele toepassingen, van machine learning tot moleculaire simulaties.

Deze toename in de vraag naar parallelle verwerking heeft geleid tot een toename van de complexiteit van GPU's. Om het ontwerp en de micro-architectuur van deze complexe chips verder te verbeteren, moeten veranderingen grondig worden gecontroleerd. Hier komen simulators in beeld. Het bouwen van een nieuwe fysieke GPU, is duur en zeer tijdrovend, waardoor een snel feedbackproces niet mogelijk is. In plaats daarvan worden simulators gebruikt om het gedrag van de chip te modelleren, zodat snel kan worden geverifieerd dat veranderingen inderdaad verbeteringen zijn.

Veel van deze simulators zijn eigendom van bedrijven en niet openbaar beschikbaar. Eén van de simulators die vaak in de academische wereld wordt gebruikt, is Accel-Sim [2]. Accel-Sim is een zeer flexibele simulator, die het mogelijk maakt om een groot aantal verschillende hardwareplatformen te simuleren. Bovendien is de simulator vrij nauwkeurig, binnen de grenzen die worden gesteld door de geheimhouding van de hardware-specificaties.

Het simuleren van een volledige workload of benchmark kan echter weken, zo niet maanden duren [3]. Om dit proces te versnellen, worden *sampling*-technieken gebruikt. Een subset van de kernels in de workload wordt geselecteerd en alleen deze kernels worden gesimuleerd. Als deze kernels correct worden gesampled en representatief zijn voor de volledige workload, kunnen de resultaten worden veralgemeend naar de volledige workload.

Enkele van deze technieken zijn *Principal Kernel Analysis* [3] en *Sieve* [4]. Op basis van profiler-data clusteren ze alle kernels in de workload. Uit elk van deze clusters wordt een representatieve kernel geselecteerd om te simuleren.

De PKA-techniek gaat nog een stap verder, door de *performance* van het eerste deel van een kernel te projecteren naar de rest van de kernel. Het idee is dat voor bijna elke kernel de IPC stabiliseert na een bepaald aantal instructies. Zodra deze stabilisatie is gedetecteerd, wordt de simulatie stopgezet en wordt aangenomen dat de rest van de kernel dezelfde IPC zal hebben.

Het nadeel van deze technieken is dat ze ervan uitgaan dat de hardware in een perfect opgewarmde toestand is. Doordat tussenliggende kernels worden weggelaten, is dit niet altijd het geval. Deze weggelaten kernels kunnen de hardware-toestand beïnvloeden, met name de caches en/of TLB's, wat op zijn beurt de prestaties van de gesimuleerde kernels beïnvloedt. Dit probleem is niet uniek voor GPU-simulatie, maar is ook aanwezig in CPU-simulatie [8], en staat bekend als het *koudestart-probleem*.

Terwijl het koudestart-probleem uitgebreid is bestudeerd in CPU-simulatie, is er weinig tot geen onderzoek gedaan naar de impact ervan op GPU-simulatie. In de context van CPU-simulatie zijn oplossingen voor het koudestart-

[1]Ghent University
[2]Department of Electronics and Information Systems, Faculty of Engineering and Architecture, Ghent University

probleem gebaseerd op het opwarmen van de datastructuren in de simulator, door het simuleren van extra instructies. Extra simulatie is echter duur, omdat simulatie veel trager is dan uitvoering in hardware. Om dit probleem in CPU-simulatie in te perken, zijn een aantal technieken voorgesteld [9, 10, 11, 12, 13, 14, 8]. Een andere benadering die is onderzocht, is het zoeken naar een correctiefactor [15], die extra simulatie vermijdt door de trace van instructies te analyseren.

Dit werk levert de volgende bijdragen:

- Het kwantificeert de impact van het koudestart-probleem op GPU-*workloads*. Hiervoor zullen we zowel hardware- als simulatieresultaten bekijken.
- Het stelt een aantal oplossingen voor om het koudestart-probleem in GPU-simulatie in te perken, geïnspireerd op de belangrijkste methoden die in CPU-simulatie worden gebruikt. We zullen deze methoden ook met elkaar vergelijken, om te zien welke het meest effectief is.

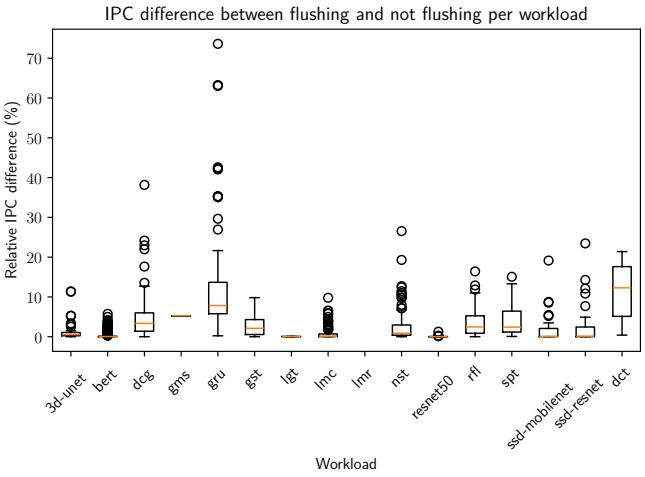
2. Het koudestart-probleem in hardware

Met behulp van NVIDIA's Nsight Compute [16] tool, waren we in staat om een aantal *workloads* te profileren[3]. Om deze *workloads* te profileren, werd een NVIDIA GeForce RTX 3080 [17] gebruikt. Deze GPU heeft 68 SM-kernen en 6 MB L2-cache.

Elke *workload* werd twee keer geprofileerd: een keer normaal, zoals het in hardware zou draaien (*noflush*), en een keer waar de caches geleegd werden tussen kernel-oproepen (*flush*). Nsight Compute ondersteunt deze optie door het `--cache-control=all` argument te gebruiken.

Hieronder vindt u een lijst van de geprofileerde *workloads*:

- De volgende *benchmarks* van de Cactus suite [18]:
  - Gromacs [19] en LAMMPS [20] (met zowel rhodo (LMR), als colloid (LMC) invoer); twee moleculaire simulatieprogramma's,
  - Gunrock [21] met zowel wegen- (GRU), als sociale netwerken (GST),
  - DCGAN, *neural style transform* (NST) [22], *reinforcement learning* (RFL), *spatial transformer* (SPT) [23] en *language translation* (LGT) van PyTorch, en
- De volgende MLCommons *benchmarks* (van hun MLPerf® Inference v2.0 collection):
  - Het ResNet50 model [24],
  - Zowel het MobileNet, als het ResNet34 variant van het SSD-model,
  - Het *Bidirectional Encoder Representations for Transformers* (BERT) [25], en
  - Het 3D U-Net model [26]
- De DCT-implementatie in de CUDA-Samples [27].



Figuur 1: IPC-verschillen in hardware

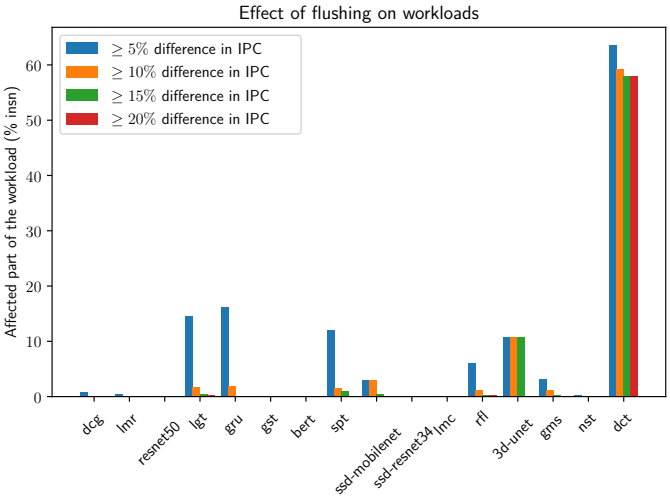
U vindt de resultaten van dit experiment in Figure 1. Deze figuur toont het relatieve verschil in IPC (relatief ten opzichte van de *noflush* IPC: $\frac{IPC_f - IPC_{nf}}{IPC_{nf}} \cdot 100\%$) voor elke *workload* die hierboven is vermeld. We merken op dat de meest veelbelovende *workloads* DCGAN en Gunrock (op wegennetwerken) van de Cactus suite zijn, samen met DCT van de CUDA-Samples.

Echter, *sampling*-methoden zoals *Sieve* [4] vertrouwen op het gewicht van een kernel; d.w.z. het aantal instructies in die kernel ten opzichte van het totale aantal instructies in de *workload*. Bovendien worden voor elke cluster die door dergelijke methoden wordt berekend, de gewichten van de kernels in die cluster opgeteld om het gewicht van het cluster te krijgen. Dat clustergewicht wordt dan gebruikt bij het generaliseren van de resultaten naar de volledige *workload*; d.w.z. er wordt een gewogen gemiddelde genomen. Wanneer we deze gewichten in aanmerking nemen, krijgen we een ander resultaat.

In Figure 2 ziet u de gewogen IPC-verschillen voor elke *workload*. Voor elke *workload* hebben we alle kernel-oproepen geïdentificeerd met minstens 5% (resp. 10%, 15% en 20%) verschil in IPC. De gewichten van al die kernels werden opgeteld, wat de hoogte van de balkjes in de grafiek bepaalt. Bijvoorbeeld, 3D-UNet heeft kernels ter waarde van ongeveer 10% van de *workload* met een IPC-verschil tussen 15% en 20%.

Met behulp van deze figuur krijgen we een iets ander beeld van de getroffen *workloads*. In de rest van dit artikel zullen we ons voornamelijk richten op de DCT- en 3D-UNet-*workloads*.

[3] Vanwege tijdgebrek werden slechts de eerste 20 000 kernels van elke workload uitgevoerd.



Figuur 2: Gewogen IPC-verschillen in hardware

2.1. Hergebruik

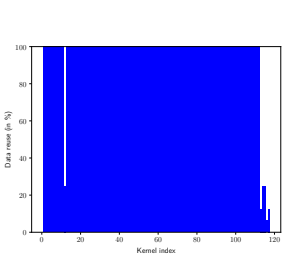
Een factor die nogal belangrijk zal blijken te zijn, is hoeveel data er tussen kernels wordt hergebruikt. We zullen ons voornamelijk richten op het voorwaartse datahergebruik; d.w.z. de data die door een kernel wordt gebruikt, en vervolgens opnieuw wordt gebruikt door de volgende. Voor een kernel k_i definiëren we het voorwaartse datahergebruik (*forward reuse*) fwd_i als:

$$fwd_i = \frac{|M_i \cap M_{i+1}|}{|M_i|} \tag{1}$$

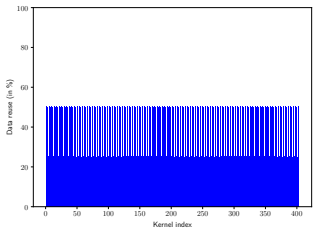
Hier is M_i de *footprint* van de kernel k_i (de verzameling unieke geheugenadressen).

In Figure 3 hebben we het voorwaartse datahergebruik gevisualiseerd voor twee *workloads*. De eerste (Figure 3a) is de DCT-implementatie die we al vermeldden, en de tweede is de *recursiveGaussian*-implementatie uit de CUDA SDK [27] (Figure 3b).

Zoals u kan zien, wordt bij DCT er veel data hergebruikt tussen kernels (consistent 100%). Aan de andere kant, de *recursiveGaussian-workload* stopt rond de 50%.



(a) Voorwaarts hergebruik bij DCT



(b) Voorwaarts hergebruik bij *recursiveGaussian*

Figuur 3: Hergebruik bij DCT en *recursiveGaussian*

3. Het koudestart-probleem in Accel-Sim

Nadat we de *workloads* in hardware hadden geprofileerd, gingen we over tot het simuleren ervan in Accel-Sim [2]. Accel-Sim is een zeer flexibele simulator, die het mogelijk maakt om een breed scala aan hardwareplatformen te simuleren. De basis van Accel-Sim is de GPGPU-Sim-simulator [28], die een *cycle-accurate simulator* is.

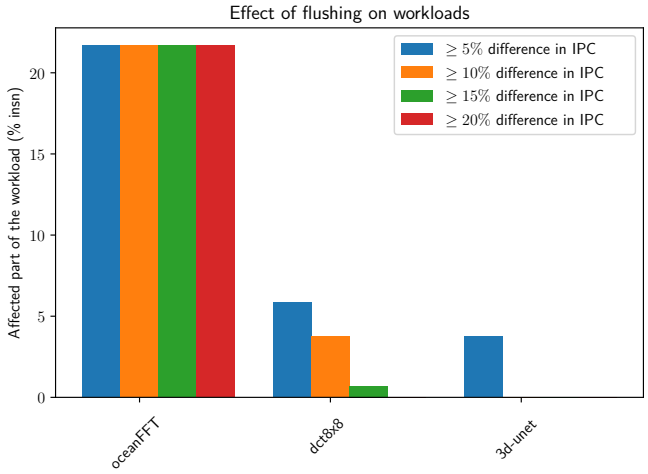
Voordat we de resultaten bespreken, eerst een woordje uitleg over de simulatoropstelling. We gebruikten de (vooraf geteste) NVIDIA GeForce RTX3070-configuratie. Enkele van de interessantere configuratieparameters worden getoond in Figure 4.

Configuration parameter	Value
L2 cache size	4 MB
Number of sets in L2 cache	64
L2 cache block size	128 B
L2 cache associativity	16
Number of memory controllers	16
Number of SMs	46
L2 Latency	187 cycles
DRAM Latency	254 cycles

Figuur 4: Configuratie-parameters bij de simulatie

We hebben elke *workload* opnieuw gesimuleerd, zoals we dat in hardware deden (zowel *flush*, als *noflush*). Accel-Sim laat ons toe om de caches te legen tussen kernel-oproepen, door de `--flush-l1 --flush-l2` argumenten te gebruiken.

We besloten om onze experimenten te beperken tot de DCT en 3D-UNet *workloads*, omdat die het meest veelbelovend waren in hardware. Een andere *workload* die veelbelovend was, was de OceanFFT-*workload* uit de CUDA SDK. We hebben elke *workload* beperkt tot 130 kernels (wat enkel een limiet is voor de 3D-UNet-*workload*), om de simulaties haalbaar te houden.



Figuur 5: Gewogen gesimuleerde IPC-verschillen

De resultaten van dit experiment kan u terugvinden in Figure 5. Net zoals in hardware, hebben we de gewogen

IPC-verschillen voor elke *workload* berekend.

Uit deze figuur kunnen we concluderen dat het koudestart-probleem veel minder ernstig is in de simulator dan in hardware. We veronderstellen dat dit komt doordat de simulator een geïdealiseerde versie van de hardware is. Sommige details zijn mogelijk verloren gegaan, wat op zijn beurt het koudestart-probleem zou kunnen hebben verzwakt.

3.1. OceanFFT

OceanFFT is een *workload* met een interessante eigenschap. Zoals u kan zien in Figure 6, zijn de IPC-waarden hoger wanneer we de caches legen.

Index	IPC (flushing)	IPC (no flushing)	Forward Reuse (%)
1	2482.26	2487.96	12.50%
2	640.58	439.84	50.00%
3	1401.63	1397.86	0.00%
4	683.88	684.10	25.00%
5	1161.11	1160.73	n/a

Figuur 6: IPC-waarden voor OceanFFT

Dit is iets wat we niet zouden verwachten; we verwachten dat de caches de uitvoering versnellen, niet vertragen. De kernel die het meest wordt beïnvloed door het koudestart-probleem is de tweede, die ongeveer 31% verschil ondervindt.

We veronderstellen dat deze kernel sneller is met geleegde caches omdat hij slechts een klein deel van de gegevens van de vorige kernel hergebruikt. De eerste kernel heeft een *footprint* van 33.5 miljoen geheugen-adressen, waarvan de tweede er slechts 4 miljoen hergebruikt (ongeveer 12.5%).

Wanneer er een groot aantal lijnen in de caches *dirty* zijn, kan dit veel *write-backs* veroorzaken. Dit kan de geheugen-pijplijn vertragen, waardoor de processors stagneren, wat de IPC verlaagt.

4. Inperkingen

We hebben een aantal oplossingen bedacht om het koudestart-probleem in Accel-Sim in te perken:

- **Perfecte opwarming:** een naïeve aanpak zou zijn om alle voorafgaande kernels te simuleren, waardoor de caches opgewarmd worden. Dit gaat echter rechtstreeks in tegen het idee van *sampling*, en vertraagt de simulatie aanzienlijk.
- **Geheugen-opwarming:** door enkel de geheugen-operaties van de voorafgaande kernels te simuleren, kunnen we de caches opwarmen. Dit is een snellere aanpak dan de vorige, die er nog steeds in slaagt om de caches op te warmen. Bovendien beperken we ons tot de meest recente kernels, omdat we verwachten dat deze het grootste effect zullen hebben, vanwege de *temporal locality*.
- **Correctiefactor:** door gebruik te maken van uitvoer van zowel *trace* als simulatie, poogden we een correctiefactor te berekenen. Deze correctiefactor bleek in staat te zijn om de nauwkeurigheid van zeer onnauwkeurige kernels te verhogen, ten koste van een iets lagere gemiddelde nauwkeurigheid.

4.1. Geheugen-opwarming

Met behulp van een aangepaste versie van de Accel-Sim *tracing*-tool, waren we in staat om een extra *trace* te genereren voor elke kernel. Deze *trace* bevat enkel de geheugen-instructies, die kunnen worden gebruikt om de caches snel op te warmen.

Uit de DCT-*workload* hebben we vier kernels gehaald die veel lijden van het koudestart-probleem. Op deze kernels hebben we de simulatie uitgevoerd met en zonder de geheugen-opwarming. We hebben deze resultaten ook vergeleken met de perfecte opwarming (waarbij elke enkele voorafgaande instructie wordt gesimuleerd), en de volledige geheugen-opwarming (waarbij alle geheugen-instructies van alle voorafgaande kernels worden gesimuleerd).

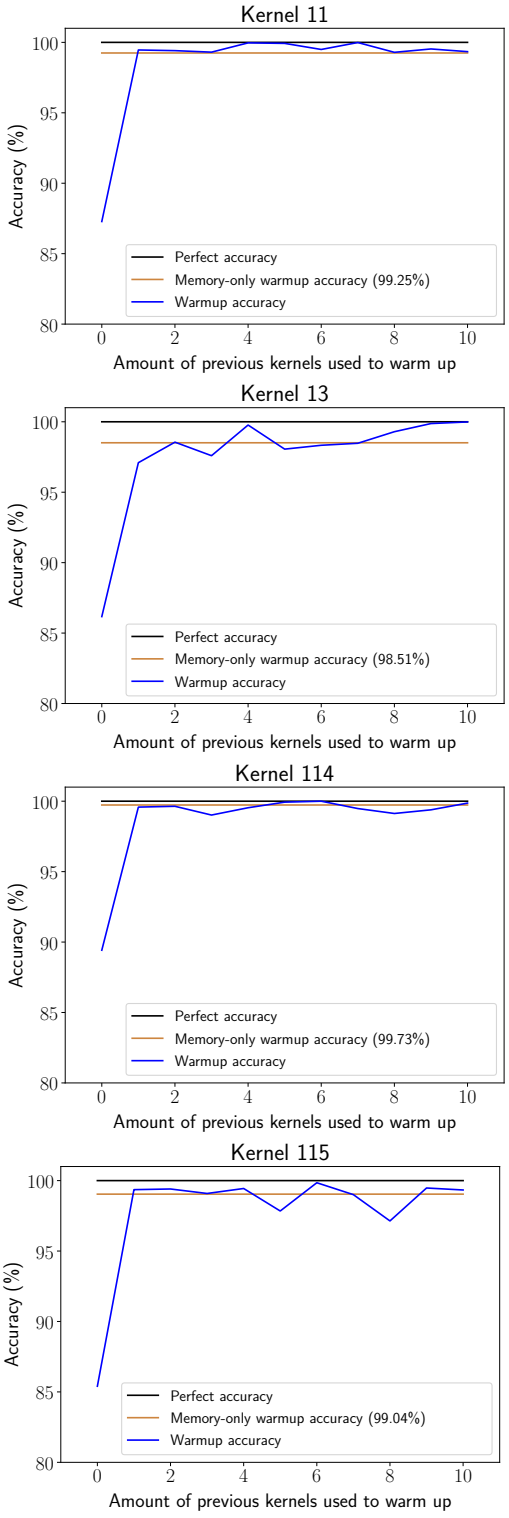
We kunnen twee grote observaties maken uit Figure 7:

1. Een enkele kernel-opwarming is meestal meer dan genoeg om de caches op te warmen; de nauwkeurigheid van alle kernels steeg tot minstens 97%.
2. Door enkel geheugen-instructies uit te voeren, is er nog steeds een verschil met de perfecte nauwkeurigheid. We veronderstellen dat dit komt door de herordening van instructies: aangezien tussentijdse instructies (bv. ALU-instructies) worden verwijderd uit de *traces*, kunnen sommige geheugen-instructies worden herordend. Dit heeft een verschillende staat van de caches als gevolg, wat de resultaten van de simulator kan beïnvloeden.

4.2. Correctiefactor

We zijn erin geslaagd om een formule te bedenken die het koudestart-probleem gedeeltelijk kan inperken, zonder extra simulatie, gebaseerd op de volgende observaties:

- We kunnen een bovengrens schatten voor de impact van het koudestart-probleem door te kijken naar het aantal *cold misses*. Een deel van deze missers is te wijten aan het koudestart-probleem, terwijl het andere deel inherent is aan de *workload*.
- Het deel van de *cold misses* dat te wijten is aan het koudestart-probleem, is evenredig met de *forward reuse*. Enkel herhaalde toegangen kunnen worden versneld door caches, dus dit zijn de enige die door het koudestart-probleem kunnen worden beïnvloed.
- Meerdere geheugenverzoeken kunnen parallel worden bediend door de *DRAM-controllers*. Hierdoor moeten we rekening houden met de bezetting van de geheugencontroller.



Figuur 7: Geheugen-opwarming bij DCT

- Het aantal verloren cycli als gevolg van het koudestart-probleem is evenredig met het verschil tussen de *DRAM latency* en de *L2 latency*.

Op basis van deze observaties hebben we de volgende formule opgesteld:

$$IPC = \frac{insn}{cycles_f - \Delta} \tag{2}$$

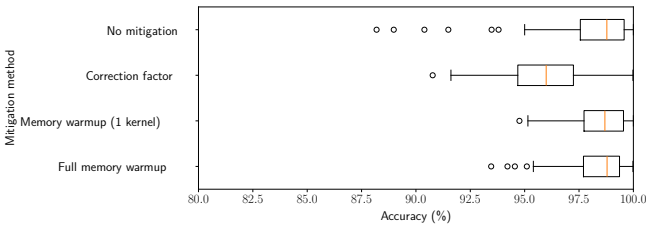
$$\Delta = \frac{accesses}{controllers \cdot line} \cdot fwd_{i-1} \cdot (DRAM - L2) \tag{3}$$

[4]

Hierbij is:

- IPC de gecorrigeerde IPC-waarde;
- $insn$ de hoeveelheid dynamisch uitgevoerde instructies;
- $cycles_f$ het aantal cycli dat de kernel nodig heeft om uit te voeren;
- Δ de correctiefactor;
- $accesses$ de hoeveelheid unique geheugenverzoeken (een bovengrens voor het aantal *cold misses*);
- $controllers$ het aantal geheugen-controllers (gewogen op hun bezetting);
- $line$ de grootte van een lijn in de cache;
- fwd_{i-1} de *forward reuse* van de *vorige* kernel; en
- $DRAM$ en $L2$ zijn de *DRAM* en *L2 latency*.

4.3. Vergelijking



Figuur 8: Vergelijking van strategieën

In Figure 8 hebben we de verschillende strategieën met elkaar vergeleken. Zoals u kan zien, zijn beide strategieën in staat om de meest getroffen kernels te verbeteren. De correctiefactor is echter minder nauwkeurig dan de geheugen-opwarming in het gemiddelde geval. De meest veelbelovende optie is de geheugen-opwarming, die in staat is om de meest getroffen kernels te verbeteren, zonder een significante computationele kost.

[4] Deze formule zal niet werken voor OceanFFT, aangezien ze het totaal aantal cycli verminderd.

5. Conclusie

In dit artikel hebben we de impact van het koudestart-probleem op GPU-*workloads* gemeten. We zijn begonnen met het analyseren van de impact in hardware, waarbij we hebben vastgesteld dat het een probleem is voor een aantal *workloads*. We hebben stilgestaan bij het belang van *forward reuse*, en zijn invloed op het koudestart-probleem.

Daarna zijn we overgegaan tot het simuleren van de *workloads* in Accel-Sim. We hebben gezien dat het koudestart-probleem veel minder ernstig is in de simulator dan in hardware, maar dat het nog steeds aanwezig is. Een onverwacht resultaat was de OceanFFT-*workload*, die sneller liep met geleegde caches.

Tenslotte hebben we enkele strategieën voor de inperking onderzocht. De meest veelbelovende hiervan is gebaseerd op extra simulatie, en kunstmatige opwarming van de caches. Deze methode slaagde erin de nauwkeurigheid drastisch te verbeteren, zonder een significante computationele kost. De andere weg die we hebben verkend, was een correctiefactor, gebaseerd op het aantal *cold misses* en de mate van datahergebruik. Deze methode was in staat om de getroffen kernels te elimineren, maar ten koste van een iets lagere gemiddelde nauwkeurigheid.

Referenties

[1] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto, Y. Shoham, J. Clark, and R. Perrault, "The ai index 2021 annual report," 2021.

[2] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accelsim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, May 2020.

[3] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal kernel analysis: A tractable methodology to simulate scaled gpu workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, ACM, Oct. 2021.

[4] M. Naderan-Tahan, H. SeyyedAghaei, and L. Eeckhout, "Sieve: Stratified gpu-compute workload sampling," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, Apr. 2023.

[5] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (gpu)," *IEEE Micro*, vol. 41, p. 42–51, Nov. 2021.

[6] NVIDIA Corporation, "Cuda," 2022.

[7] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?," *Queue*, vol. 6, p. 40–53, Mar. 2008.

[8] L. Eeckhout, Y. Luo, K. De Bosschere, and L. K. John, "Blrl: Accurate and efficient warmup for sampled processor simulation," *The Computer Journal*, vol. 48, no. 4, pp. 451–459, 2005.

[9] G. Lauterbach, "Accelerating architectural simulation by parallel execution of trace samples," in *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 1, pp. 205–210, 1994.

[10] R. Kessler, M. Hill, and D. Wood, "A comparison of trace-sampling techniques for multi-megabyte caches," *IEEE Transactions on Computers*, vol. 43, no. 6, pp. 664–675, 1994.

[11] T. Conte, M. Hirsch, and K. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pp. 468–477, 1996.

[12] T. Conte, M. Hirsch, and W.-M. Hwu, "Combining trace sampling with single pass methods for efficient cache simulation," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 714–720, 1998.

[13] J. Haskins and K. Skadron, "Minimal subset evaluation: rapid warm-up for simulated hardware state," in *Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors. ICCD 2001*, pp. 32–39, 2001.

[14] J. Haskins and K. Skadron, "Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation," in *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.*, pp. 195–203, 2003.

[15] D. A. Wood, M. D. Hill, and R. E. Kessler, "A model for estimating trace-sample miss ratios," in *Proceedings of the 1991 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, SIGMETRICS91, ACM, Apr. 1991.

[16] NVIDIA Corporation, "NVIDIA nsight compute."

[17] NVIDIA Corporation, "NVIDIA ampere ga102 gpu architecture," whitepaper, 2021.

[18] M. Naderan-Tahan and L. Eeckhout, "Cactus: Top-down gpu-compute benchmarking using real-life applications," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, Nov. 2021.

[19] S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, and E. Lindahl, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs," *The Journal of Chemical Physics*, vol. 153, Oct. 2020.

[20] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[21] Y. Wang, Y. Pan, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liu, A. T. Riffel, and J. D. Owens, "Gunrock: GPU graph analytics," *ACM Transactions on Parallel Computing*, vol. 4, pp. 3:1–3:49, Aug. 2017.

[22] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *ArXiv*, vol. abs/1508.06576, 2015.

[23] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *ArXiv*, vol. abs/1506.02025, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[26] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2016.

[27] NVIDIA Corporation, "CUDA samples."

[28] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE, Apr. 2009.