базових технік інкапсуляції та безперешкодно пропускають трафік тестових DNS-тунелів. Цей факт підтверджує наявність вразливостей і обґрунтовує необхідність глибокого моделювання методів обходу фільтрації, для чого проєктований протокол виступатиме основним інструментом тестування.

Попередній аналіз підтверджує, що традиційні DPI-рішення провайдерів, орієнтовані на сигнатурний пошук, є критично вразливими до методів динамічної обфускації та нестандартного шифрування на рівні DNS. Для надійного захисту інфраструктури та закриття цих "сліпих зон" необхідно змістити фокус із прямої інспекції корисного навантаження на евристичні моделі. Ефективна протидія сучасним прихованим каналам зв'язку потребує впровадження систем поведінкового аналізу на базі машинного навчання [6], здатних своєчасно виявляти аномалії та нетипові патерни DNS-комунікації.

1. Duan R., Liu D. Understanding DNS Tunneling Traffic in the Wild. Unit 42, Palo Alto Networks. 2023. URL: https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild/
2. Decoy Dog Is No Ordinary Pupy – Infoblox Reveals Shift in Malware Tactics After Initial Discovery. Infoblox. 2023. URL: https://www.infoblox.com/news/news-events/press-releases/decoy-dog-is-no-ordinary-pupy-infoblox-reveals-shift-in-malware-tactics-after-initial-discovery/
3. Mayer D. ChamelGang and ChamelDoH: A DNS-over-HTTPS implant. Stairwell Threat Research. 2023. URL: https://blog.netmanageit.com/content/files/2023/06/Report-ChamelGang-and-ChamelDoH-A-DNS-over-HTTPS-implant.pdf
4. Amirov N., Isik B., Tuncer B. I., Bahtiyar S. DNS Tunneling: Threat Landscape and Improved Detection Solutions. arXiv preprint arXiv:2507.10267. 2025. URL: https://arxiv.org/abs/2507.10267
5. Salat L., Davis M., Khan N. DNS Tunnelling, Exfiltration and Detection over Cloud Environments. Sensors. 2023. Vol. 23, №5. P. 2760. DOI: 10.3390/s23052760
6. Ali F., Afaq M., Niazi M., Behzad M. From Graphs to Gates: DNS-HyXNet, A Lightweight and Deployable Sequential Model for Real-Time DNS Tunnel Detection. arXiv preprint arXiv:2512.09565. 2025. URL: https://arxiv.org/abs/2512.09565

GPU-Adapted Compact Hashing with Bitonic Sort for Neighborhood Search in SPH

UDC 004.94 (043.2)

Ostap Hrytsyshyn[1], Valeriy Trushevskyy[2]

Ivan Franko National University of Lviv, [1]ostap.hrytsyshyn@lnu.edu.ua, [2]valeriy.trushevsky@lnu.edu.ua

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian particle-based method widely used for simulating fluid dynamics. Each particle interacts only with neighbors

located within the kernel support radius h. A naive search over all particle pairs requires $O(n^2)$ operations, which becomes prohibitive for large n, making efficient neighborhood search one of the most important performance bottlenecks of any practical SPH solver [1].

Uniform grids and spatial hashing reduce the cost to approximately O(n), but on GPU architectures additional considerations are required: memory access patterns must be coalesced, dynamic memory allocation should be avoided, and parallel sorting algorithms must be carefully chosen. This paper presents a GPU-adapted compact hashing scheme combined with Bitonic Sort, designed to maintain memory locality and predictable runtime for real-time SPH simulations [2, 3].

Compact Hashing. Following the approach of Ihmsen et al. [2], a uniform grid is constructed over the simulation domain with cell size equal to the kernel support radius h. Each particle is mapped to a grid cell based on its position. The 3D-to-1D hash function is given by:

$$c = (\lfloor x/d \rfloor p_1 \oplus \lfloor y/d \rfloor p_2 \oplus \lfloor z/d \rfloor p_3) \bmod m, \quad (1)$$

where d is the cell size, p_1, p_2, p_3 are large prime numbers, and m is the hash table size. To ensure predictable memory access and avoid runtime overhead from dynamic allocation, the hash table is pre-allocated with size equal to the number of particles. While this does not eliminate hash collisions, it minimizes allocation overhead and enables efficient parallel processing.

Bitonic Sort on GPU. After hash indices are computed, particles must be sorted by their grid index to ensure spatial locality during neighbor search. For GPU execution we adopt Bitonic Sort introduced by Batcher [4], which is highly structured and well-suited to parallel architectures. A sequence is called bitonic if it first increases and then decreases. The algorithm recursively builds bitonic subsequences and merges them into a sorted result using compare-and-swap operations executed in parallel by GPU threads.

The overall complexity is $O(n \log^2 n)$ but each level executes fully in parallel, which gives a wall-clock advantage when n is large. After sorting, particles in the same or adjacent grid cells are stored contiguously in memory, dramatically improving cache utilization during the neighbor query step.

Neighborhood Query. For each particle, only the 27 (or 9 in 2D) cells in the immediate vicinity of its grid cell are evaluated. The cost of the query scales linearly with the number of particles, i.e. O(n). The pseudocode of the integrated procedure is presented in Algorithm 1.

Algorithm 1. GPU-adapted neighborhood search

```
1: for all particles i in parallel do
    2:   compute hash c_i from particle position x_i via Eq. (1)
3: end for
4: BitonicSort(particles by c_i)  // parallel on GPU
5: for all particles i in parallel do
    6:   for all 27 cells in vicinity of c_i do
        7:     collect candidates with ||x_j − x_i|| < h
    8:   end for
9: end for
```

Results. The proposed scheme was implemented using compute shaders and tested on a benchmark with up to 75,000 fluid particles. The GPU-based implementation completes one full simulation step in approximately 10 ms. For comparison, a multi-threaded CPU implementation requires about 250 ms, and a single-threaded CPU implementation requires about 690 ms for an equivalent particle count. The reported speedup factor exceeds 25× compared to the multi-threaded CPU baseline and 65× compared to the single-threaded baseline, confirming that the dominant share of the gain comes from the parallel neighborhood search [5].

Pre-allocation of the hash table with size equal to the number of particles eliminates allocation overhead and stabilizes per-step timings. The locality enforced by sorting reduces irregular memory access in the subsequent density and pressure passes, which is particularly important for IISPH-based incompressible solvers where multiple Jacobi iterations are executed per step.

Conclusions. This paper presented a GPU-adapted neighborhood search procedure for SPH simulations based on compact hashing and Bitonic Sort. The combination of pre-allocated hash tables, structured parallel sorting and cell-based neighbor queries enables linear scaling with the number of particles and real-time performance for ensembles of tens of thousands of particles. Future work includes adaptive hash table sizing and extension to multi-GPU configurations.

1. Monaghan J. J. Smoothed particle hydrodynamics. Reports on Progress in Physics. – 2005. – Vol. 68. – P. 1703–1759.
2. Ihmsen M., Akinci N., Becker M., Teschner M. A parallel SPH implementation on multi-core CPUs. Computer Graphics Forum. – 2011. – Vol. 30, No. 1. – P. 99–112.
3. Ihmsen M., Cornelis J., Solenthaler B., Horvath C., Teschner M. Implicit incompressible SPH. IEEE Transactions on Visualization and Computer Graphics. – 2014. – Vol. 20, No. 3. – P. 426–435.
4. Batcher K. E. Sorting networks and their applications. Proceedings of the AFIPS Spring Joint Computer Conference. – 1968. – Vol. 32. – P. 307–314.
5. Hrytsyshyn O., Trushevskyy V. Application of IISPH for incompressible fluid dynamics simulation. Вісник Львівського університету. Серія прикладна математика та інформатика. – 2024. – Вип. 33. – С. 55–68.

Автоматизація процесів інтеграції та розгортання вебзастосунків

УДК 004.42

Петришин Ярослав[1], Мудрик Іван[2]

Тернопільський національний технічний університет імені Івана Пулюя,
[1]aroslav_petryshyn1608@tntu.edu.ua, [2]i1mudryk@tntu.edu.ua

Сучасні вебзастосунки відрізняються складною багаторівневою архітектурою, яка включає серверну частину, клієнтський застосунок, бази даних та сервіси реального часу. За таких умов ручне тестування й розгортання програмного продукту стає неефективним і ризикованим. Методологія CI/CD (Continuous Integration / Continuous Deployment) забезпечує автоматизацію процесів перевірки, складання та доставки змін до виробничого середовища,