Aceleração de Simulações de Vida Computacional Emergente com CUDA: Experimentos com o Simulador CuBFF

Luiz C.S. Almeida[1], **Wellington S. Martins**[1]

[1] Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 74.690-900 – Goiânia – GO – Brazil

luizalmeida@inf.ufg.br, wsmartins@ufg.br

Abstract. *Recent research has elevated the study of artificial life to a new level by demonstrating that self-replicating programs can emerge spontaneously, without any prior design or selection. In other words, self-replication can arise naturally from simple interactions between code fragments in purely computational environments — a discovery that redefines the concept of "digital life". However, empirically proving these phenomena requires high computational power, as the simulations involve billions of independent and stochastic executions, in which programs interact, modify each other, and give rise to new patterns. To efficiently explore this vast state space and apply complex metrics of entropy and compression, the use of parallel processing becomes indispensable. In this work, we present our experiments with the CUDA version of the CuBFF simulator and discuss strategies to further optimize performance, reducing communication costs and improving GPU utilization in the study of the emergence of computational life.*

Resumo. *Pesquisas recentes elevaram o estudo da vida artificial a um novo patamar ao demonstrarem que programas autorreplicantes podem emergir espontaneamente, sem qualquer projeto ou seleção prévia. Em outras palavras, a autorreplicação pode surgir naturalmente de interações simples entre fragmentos de código em ambientes puramente computacionais — uma descoberta que redefine o conceito de "vida digital". Entretanto, comprovar empiricamente esses fenômenos exige alto poder computacional, pois as simulações envolvem bilhões de execuções independentes e estocásticas, em que programas interagem, se modificam e dão origem a novos padrões. Para explorar de forma eficiente esse vasto espaço de estados e aplicar métricas complexas de entropia e compressão, torna-se indispensável o uso de processamento paralelo. Neste trabalho, apresentamos nossos experimentos com a versão CUDA do simulador CuBFF e discutimos estratégias para otimizar ainda mais o desempenho, reduzindo custos de comunicação e melhorando o aproveitamento das GPUs no estudo da emergência da vida computacional.*

1. Introdução

Dialogando com clássicos como o Jogo da Vida [Gardner 1970] e os estudos de Langton sobre vida artificial [Langton 1986], o artigo de Agüera y Arcas et al. [Agüera y Arcas et al. 2024] investiga a emergência espontânea de autorreplicadores a partir de programas aleatórios, um estudo de vasto escopo que explora diferentes substratos computacionais e simulações espaciais para entender a origem da vida. Dada a

complexidade e a necessidade de processar trilhões de interações, o trabalho depende de uma arquitetura de paralelização massiva, sendo este o foco da presente análise. Este artigo analisa em detalhe a implementação em CUDA (Compute Unified Device Architecture) do motor de simulação [Paradigms of Intelligence 2024], destacando as estratégias de paralelismo que possibilitam a execução massivamente acelerada dos experimentos em GPU, permitindo a interação simultânea de centenas de milhares de programas e tornando viável a observação de fenômenos emergentes em larga escala. Apresentamos também resultados de experimentos conduzidos localmente com essa implementação e validamos o surgimento de vida computacional mesmo com poucas iterações.

2. Simulador cubff

A simulação modela interações como reações químicas ($A + B \rightarrow A' + B'$), permitindo que autorreplicadores (S) surjam de reações autocatalíticas ($S + F \rightarrow 2 \cdot S$). Para detectar a emergência de ordem a partir do caos, o artigo original introduz a métrica "entropia de alta ordem", definida como a diferença entre a entropia de Shannon (aleatoriedade) e a complexidade de Kolmogorov (estrutura, aproximada pela compressão Brotli). No entanto, observar a emergência de autorreplicadores requer a simulação de uma vasta "sopa primordial" com centenas de milhares de programas interagindo, uma tarefa computacionalmente cara para CPUs tradicionais, tornando a implementação massivamente paralela uma dependência crítica.

A eficiência da simulação deriva da divisão do trabalho em múltiplos kernels CUDA especializados. O processo inicia-se com o kernel `InitPrograms`, onde cada thread da GPU é encarregada de inicializar um único programa de 64 bytes com dados aleatórios, gerando em paralelo massivo a população inicial de 2^{17} (131.072) programas. A reprodutibilidade é garantida pelo gerador SplitMix64, usando semente e índice da thread para criar uma população única.

O núcleo da simulação reside no kernel `MutateAndRunPrograms`, executado a cada época (geração) e implementando a dinâmica de interação. O processo inicia-se com a seleção e concatenação, onde cada thread escolhe dois programas com base em índices pré-embaralhados, e os concatena em uma fita de memória local de 128 bytes. Em seguida, a mutação é aplicada: cada byte da fita concatenada possui uma baixa probabilidade (padrão: 0,024%) de ser substituído por um valor aleatório.

Após a mutação, a execução é invocada: a máquina virtual BFF (função 'Language::Evaluate') opera sobre a fita de 128 bytes por um número máximo de instruções (ex: 8.192). Finalmente, na separação e escrita, a fita modificada é dividida em dois programas "filhos" (A' e B') de 64 bytes, que são escritos de volta nas posições originais dos pais na memória global. Para otimizar este ciclo, que é o gargalo da simulação, primitivas de redução no nível do warp agregam contagens de instruções, e a configuração da grid (blocos de 32 threads) é alinhada à arquitetura NVIDIA para máxima eficiência.

Periodicamente, um terceiro kernel, `CheckSelfRep`, é lançado para identificar a existência de autorreplicadores, uma tarefa computacionalmente intensiva. Cada thread testa um programa candidato concatenando-o com 13 sequências de "ruído" (programas aleatórios) diferentes. O programa é executado, e seu resultado é usado como entrada para a próxima "geração" (concatenado com mais ruído), em um processo repetido por quatro gerações sucessivas. Um programa é classificado como autorreplicador se mantiver sua

integridade estrutural (consistência de bytes) em pelo menos 25% dessas iterações.

3. Experimentos

Realizamos a simulação em uma GPU Nvidia RTX 4060 Ti (arquitetura `sm_89`). A execução foi configurada com 100.000 épocas, intervalo de impressão de 5.000 e semente 42. Utilizamos a linguagem `bff_noheads` (variação de Brainfuck [Müller 1993]) uma linguagem minimalista de 10 instruções (<, >, {, }, −, +, ., ,, [,]) onde código e dados coexistem no mesmo espaço de memória. Esta característica permite a auto-modificação e confere robustez a mutações aleatórias, pois apenas 10 dos 256 valores de bytes são instruções válidas. A implementação CUDA, processando $\approx 10^9$ instruções/segundo, foi crucial para observar as transições de fase em menos de 16.000 épocas, com autorreplicadores emergindo consistentemente.

Os dados confirmam a transição de fase: a entropia de alta ordem e o número de autorreplicadores, iniciando em -0,000249 e 0 (época 1), saltam para 5,05 e 32.681 na época 10.001, e estabilizam em $\approx 6,34$ e ≈ 101.913 ao final das 100.001 épocas. Paralelamente, iniciamos uma análise da implementação CUDA. Esta análise, a ser refinada em trabalhos futuros, identificou alguns possíveis gargalos, focando especialmente na função de avaliação de replicadores `CheckSelfRep`, que demonstra uma baixa ocupação da GPU (menos de 10%) devido ao "spill" de memória local (1664 bytes/thread).

4. Conclusão

O trabalho de Agüera y Arcas et al. demonstra de forma convincente que a vida computacional pode emergir espontaneamente em substratos simples através de automodificação. Nosso experimento, com 100.000 épocas, confirmou a transição de fase prevista, mostrando um aumento rápido e correlacionado da entropia de alta ordem e do número de autorreplicadores, que, após as primeiras iterações, estabilizaram em valores característicos de um estado organizado e autorreplicante.

Trabalhos futuros focarão em ampliar experimentos para validação e otimizar o kernel `CheckSelfRep`, onde um redesign reduzindo movimentações de memória promete *speedup* considerável.

Referências

Agüera y Arcas, B., Alakuijala, J., Evans, J., Laurie, B., Mordvintsev, A., Niklasson, E., Randazzo, E., and Versari, L. (2024). Computational Life: How Well-formed, Self-replicating Programs Emerge from Simple Interaction. *arXiv e-prints*.

Gardner, M. (1970). Mathematical games. *Scientific American*, 223(4):120–123.

Langton, C. G. (1986). Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1):120–149. Proceedings of the Fifth Annual International Conference.

Müller, U. (1993). Brainfuck compiler. `https://aminet.net/package/dev/lang/brainfuck-2`.

Paradigms of Intelligence (2024). Cubff: Cuda-based implementation of a self-modifying soup of programs. `https://github.com/paradigms-of-intelligence/cubff`.