Estudo preliminar de métricas de produtividade e portabilidade para linguagens de programação paralela*

Daniela F. Daniel[1], **Jairo Panetta**[1]

[1]Divisão de Ciência da Computação – Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos – São Paulo – Brasil

{danieladaniel2004, jairo.panetta}@gmail.com

Abstract. *The variety of parallel programming languages poses the challenge of selecting an appropriate language for each target platform. Using classical and recent metrics, this preliminary study compares parallel programming languages with respect to programming productivity and performance portability. We used three parallel versions of the Game of Life coded in OpenMP, CUDA and Kokkos. Results show that OpenMP and Kokkos require less coding effort than CUDA, that OpenMP is the best choice for running on the CPU and that Kokkos is the best choice for the GPGPU.*

Resumo. *A variedade de linguagens de programação paralela impõe o desafio de escolher uma linguagem apropriada para cada plataforma alvo. Utilizando métricas clássicas e recentes, este estudo preliminar compara linguagens de programação paralela com relação à produtividade de programação e à portabilidade efetiva do código. Utilizamos três versões paralelas do Jogo da Vida escritas em OpenMP, CUDA e Kokkos. Resulta que OpenMP e Kokkos requerem menor esforço de codificação do que CUDA, que OpenMP é a melhor escolha para execução na CPU e Kokkos é a melhor escolha para a GPGPU.*

1. Introdução

Existem várias linguagens de programação disponíveis para paralelizar códigos sequenciais, seja em arquiteturas *multicore* ou *manycore*. É possível escolher entre linguagens que usam mecanismos distintos para expressar paralelismo: diretivas de compilação, extensões de linguagens tradicionais ou bibliotecas de procedimentos. Dada a variedade de linguagens e a evolução constante do hardware, é pertinente avaliar o esforço inerente ao desenvolvimento de programas paralelos e o tempo de execução atingido, com o objetivo de medir a facilidade de uso, a portabilidade e o desempenho das opções existentes.

Neste trabalho, aplicamos métricas clássicas e métricas recentes a um programa paralelo, o Jogo da Vida, a fim de comparar as linguagens de programação paralela usadas para escrevê-lo. As próximas seções desse artigo estão organizadas da maneira descrita a seguir.

Na seção 2 são descritas métricas para medir o esforço humano no desenvolvimento de software de alto desempenho. A seção 3 apresenta uma métrica de portabilidade efetiva da aplicação para um conjunto de plataformas de hardware. Na seção 4, as métricas são aplicadas em diferentes implementações do Jogo da Vida. Por fim, as conclusões e perspectivas para trabalhos futuros são apresentadas na seção 5.

*Este trabalho foi parcialmente financiado pelo Termo de Cooperação 0050.0102253.16.9 entre a Petrobras e a Universidade Federal do Rio Grande do Sul

2. Esforço de codificação e produtividade

É opinião generalizada dos usuários de HPC de que OpenMP é mais fácil de usar do que MPI, mas há poucas evidências empíricas que validem esse consenso. Em 2005, frente a escassez de estudos dedicados a medir o esforço dos programadores em escrever código paralelo, [Hochstein et al. 2005] realizaram experimentos com alunos de cursos introdutórios de HPC visando medir a influência da adoção de OpenMP ou MPI no esforço de desenvolvimento. O esforço foi medido pelo tempo de codificação (em pessoa-hora) e pelo tempo de codificação por linha de código modificada. Concluíram que MPI requer mais esforço de desenvolvimento do que OpenMP, tanto em tempo absoluto quanto em tempo por linha de código.

Maior esforço não significa, obrigatoriamente, menor produtividade, pois medir produtividade requer considerar a qualidade do resultado atingido. Embora os autores tenham medido a qualidade das soluções dos estudantes pelo *speedup* atingido, i.e., a razão entre o tempo de execução da versão paralela e da versão serial, o trabalho não combinou essas duas métricas. O uso conjunto dessas grandezas ocorreu ainda em 2005, quando estudos conduzidos por [Funk et al. 2005] usaram o *speedup* para desenvolver uma métrica de *produtividade* a partir de uma medida de esforço relativo:

$$Produtividade = \frac{Speedup}{Esforço\ Relativo}$$

onde

$$Esforço\ Relativo = \frac{Esforço\ Paralelo}{Esforço\ Serial}$$

Os autores apresentaram diversas formas de medir o esforço relativo. A forma mais direta é medir o tempo de codificação das soluções paralela e serial. Contudo, tempo de desenvolvimento é difícil de se obter, mesmo em ambientes instrumentados e controlados. Além disso, fatores como a experiência do programador e sua familiaridade com uma linguagem paralela tendem a favorecer os resultados dessa linguagem em detrimento de outras.

A alternativa adotada foi usar o número de linhas de código (LOC) como medida de esforço. Nessa abordagem, o valor do esforço relativo corresponde ao fator de expansão de código utilizado por [Hochstein et al. 2005], se adotada uma versão serial comum que sirva como base para as implementações paralelas.

Estudos mais recentes também usam o número de linhas de código como medida de esforço. Por exemplo, [Memeti et al. 2017] definem esforço de paralelização como o percentual das linhas de código utilizadas em construções paralelas em relação ao número de linhas de código total da aplicação:

$$Effort_{par}\,[\%] = 100 * LOC_{par}/LOC_{total}$$

3. Portabilidade efetiva

A recente popularização de aceleradores (como as GPGPUs) ampliou a importância da portabilidade de programas paralelos, pois o desenvolvimento e a manutenção de uma aplicação para cada hardware novo incorrem em custos significativos. Nos últimos anos,

surgiram novas linguagens de programação (como OpenACC) e extensões de linguagens tradicionais (como OpenMP) cujo principal objetivo é a portabilidade, permitindo que um mesmo código fonte seja compilado para diferentes alvos mantendo resultados aceitáveis de desempenho.

Nesse contexto, o termo "portabilidade efetiva" (em inglês, *performance portability*) é utilizado na literatura como um indicativo do grau de versatilidade das linguagens paralelas. Em 2017, [Pennycook et al. 2017] propuseram uma métrica para medir a portabilidade efetiva usando duas funções de eficiência:

(1) a *eficiência arquitetural* corresponde ao percentual atingido em relação à performance de pico de determinado hardware, seja em largura de banda ou FLOP/s;
(2) a *eficiência da aplicação* é o percentual atingido em relação ao melhor resultado obtido num dado hardware.

Tais eficiências variam com o hardware utilizado. Seja H um conjunto de plataformas de interesse e $|H|$ a cardinalidade de H. Seja $e(a, p)$ uma função de eficiência da aplicação a em resolver o problema p. O trabalho define a portabilidade efetiva P como a média harmônica das eficiências em H:

$$P = \begin{cases} \dfrac{|H|}{\sum_{i \in H} \frac{1}{e_i(a,\, p)}} & \text{se } i \text{ é suportado } \forall i \in H \\ 0 & \text{senão} \end{cases}$$

4. O Jogo da Vida: implementações e resultados

Paralelizar o Jogo da Vida [Conway 1970] é exercício comum em cursos introdutórios de HPC. A partir de uma versão serial escrita em linguagem C, codificamos o Jogo da Vida em OpenMP, CUDA e *Kokkos* [Edwards et al. 2014]. Este último é uma biblioteca de *templates* escrita em C++ que permite compilar um mesmo código fonte para CPUs (gerando OpenMP ou *Ptheads*) e para GPGPUs (gerando CUDA). Como compiladores completos para OpenMP 4.5 ainda não estão disponíveis na máquina alvo, não testamos a portabilidade de programas OpenMP para CPUs e GPGPUs. Os códigos foram executados em um nó do computador [SDumont], composto por dois processadores *Intel Xeon E5-2695 v2* e duas GPGPUs *NVIDIA Tesla K40*.

As versões OpenMP e *Kokkos OpenMP* executaram nas CPUs com 16 *threads*. As versões CUDA e *Kokkos CUDA* executaram em uma GPGPU. A contagem de linhas (LOC) desconsidera as linhas em branco e os comentários. A versão serial possui 116 linhas e o tempo de execução foi de 13,57 segundos (4084 iterações). Esses valores foram usados para calcular o *Esforço Relativo* e a *Produtividade*. A Tabela 1 mostra os resultados.

A eficiência da aplicação escrita em *Kokkos* foi de 46,18% na CPU e 100%, na GPGPU, resultando em portabilidade P de 63,18%, já que $H = \{CPU,\ GPGPU\}$ e $|H| = 2$.

5. Discussões e conclusões

Usamos as três versões paralelizadas do Jogo da Vida para medir a portabilidade efetiva e a produtividade de programação a partir de uma versão serial comum. Como se trata de

Tabela 1. Resultados do Jogo da Vida num tabuleiro de 1024x1024

Aplicação	LOC	*Esforço Relativo*	Tempo (s)	*Speedup*	*Produtividade*
OpenMP	127	1,09	1,51	9,00	8,22
CUDA	147	1,27	1,13	11,98	9,46
Kokkos OpenMP	125	1,08	3,27	4,14	3,85
Kokkos CUDA	125	1,08	1,02	13,35	12,39

um problema simples, o esforço relativo em todas as versões foi pequeno, mas a diferença entre os valores das versões em CUDA e OpenMP reforça a noção de que OpenMP é mais fácil de usar.

Kokkos apresentou o menor esforço relativo, porém o código compilado para OpenMP foi duas vezes mais lento na CPU do que o código OpenMP puro, penalizando a produtividade e a portabilidade efetiva. Na GPGPU, no entanto, o menor tempo de execução do Jogo da Vida veio da implementação em *Kokkos*, sugerindo que *Kokkos* está num nível de dificuldade intermediário entre CUDA e OpenMP.

Espera-se que em trabalhos futuros uma quantidade maior de problemas e de linguagens sejam testados. Outro ponto a ser explorado é a variação que o tamanho do problema exerce sobre os valores de produtividade e portabilidade. Também se espera avaliar a viabilidade de melhorar os resultados às custas de um maior esforço de codificação, na intenção de testar a sensibilidade das métricas a códigos otimizados.

Referências

Conway, J. (1970). Game of life. *Scientific American*, 223:120–123.

Edwards, H. C., Trott, C. R., and Sunderland, D. (2014). Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216.

Funk, A., Basili, V., Hochstein, L., and Kepner, J. (2005). Application of a development time productivity metric to parallel software development. *Proceedings of the second international workshop on Software engineering for high performance computing system applications - SE-HPCS 05*.

Hochstein, L., Carver, J., Shull, F., Asgari, S., Basili, V., Hollingsworth, J., and Zelkowitz, M. (2005). Parallel programmer productivity: A case study of novice parallel programmers. *International Conference for High Performance Computing, Networking and Storage (SC'05)*.

Memeti, S., Li, L., Pllana, S., Kolodziej, J., and Kessler, C. (2017). Benchmarking *OpenCL*, *OpenACC*, *OpenMP*, and *CUDA*: Programming productivity, performance, and energy consumption. *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing – ARMS-CC 17*.

Pennycook, S., Sewall, J., and Lee, V. (2017). Implications of a metric for performance portability. *Future Generation Computer Systems*.

SDumont. Sistema brasileiro de computação petaflópica. `http://www.sdumont.lncc.br/machine.php`. Acesso: 30-01-2018.