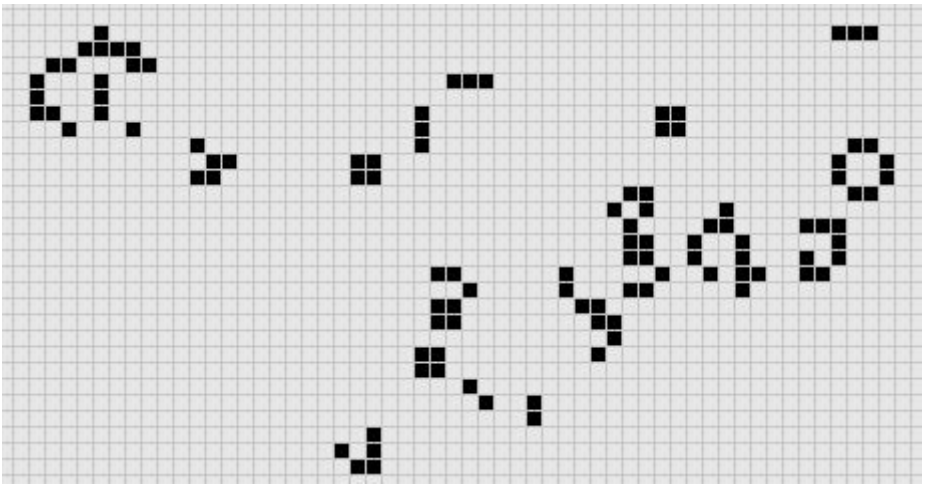
Relatório de Avaliação de Desempenho

Simulação do Jogo da Vida

Aluno: Marcelo Padilha Fontes de Barros
Professor: Ricardo Vargas Dorneles



A simulação escolhida para o teste de avaliação de desempenho foi o autômato celular Jogo da Vida de Conway. Essa é uma simulação que se apresenta em uma área em grade, teoricamente infinita, onde regras simples de interação entre as células geram complexidade no sistema. Cada célula interage com células ao seu redor, e as regras são como se segue:

1. Uma célula viva com apenas uma célula vizinha viva ou nenhuma morre (solidão).
2. Uma célula viva com quatro ou mais células vizinhas vivas morre (superpopulação).
3. Uma célula viva com dois ou três vizinhos sobrevive.
4. Uma célula vazia com exatamente três vizinhos se torna viva (reprodução).

O desenvolvimento do programa

- A área criada no programa desenvolvido para o teste de desempenho é uma matriz principal 20x50 (para visualização na tela e por limitação de testes com variável local e não dinâmica).
- Cada posição dentro da matriz principal representa a localização de uma célula da simulação que pode estar viva ou morta. Coordenadas iniciais são dadas para criar as células que darão início na simulação.
- Todas as células da matriz são percorridas e cada uma delas recebe o valor do número de vizinhos vivos que possui em suas oitos direções.
- Uma matriz auxiliar é usada para projetar o resultado das regras do autômato para cada célula percorrida na matriz principal.
- Depois de todas mudanças serem passadas para a matriz temporária, ela é copiada para a matriz principal. Isso marca o final do turno e a apresentação do resultado na tela.
- O processo se reinicia e continua consecutivamente em um total de duzentos mil turnos (200.000 iterações).

Apesar do programa apresentar a visualização de toda a interação da simulação na tela para o usuário, ela precisa ser desabilitada para os testes de desempenho, visto que a impressão na tela e o processo “usleep”, que mantém o frame suspenso para ser captado pelo olho humano, criam ambos interrupção no processamento e interferem nos resultados.

Quatro versões da simulação foram criadas:

Game of Life: Alocação de memória dinâmica das matrizes principal e auxiliar (uso de memória). `int **matriz = new int *[linhas];`

```
for(int i=0; i<linhas; i++){
      matriz[i] = new int [colunas];
}
```

Game of Life 2: Alocação de memória dinâmica das matrizes principal e auxiliar, mas a busca nas matrizes são realizadas por colunas.

```
for(int j=0; j<colunas-1; j++){
        for(int i=0; i<linhas-1; i++){
```

Game of Life 3: Inicialização local das matrizes principal e auxiliar (uso de pilha).

```
int matriz[linhas][colunas];
```

Game of Life 4: Criação de vetores como representação lógica da matriz principal e auxiliar onde as buscas são realizadas da seguinte forma:

```
matriz[i*colunas+j]
```

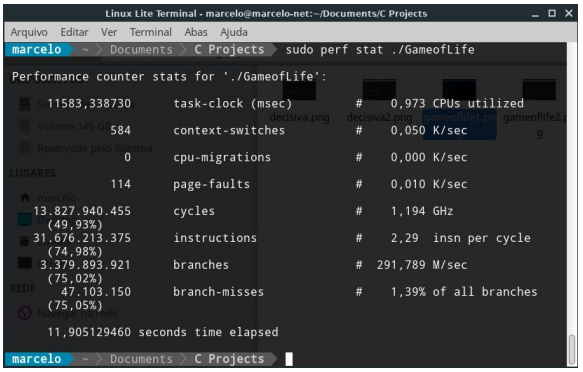
Resultados e Conclusões

O primeiro teste realizado foi o teste de velocidade de conclusão da tarefa. A média de tempo de cada uma das versões é como se segue:

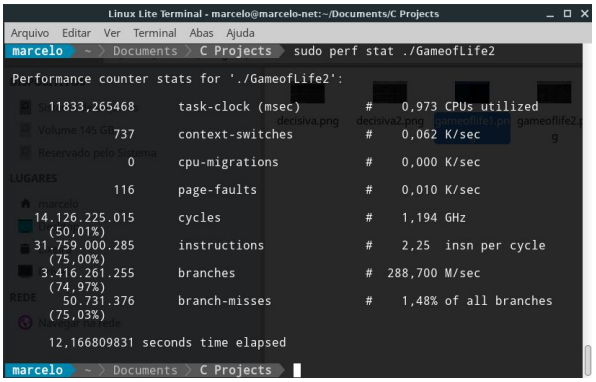
Game of Life: 12,234
Game of Life 2: 12,542
Game of Life 3: 13,309
Game of Life 4: 11,208

A versão onde as matrizes de duas dimensões são representadas logicamente por vetores apresentou-se a mais rápida, enquanto que a versão onde as matrizes foram inicializadas na pilha foi a versão mais lenta. A razão para essa diferença de tempo da tarefa é revelada nos testes que seguem.

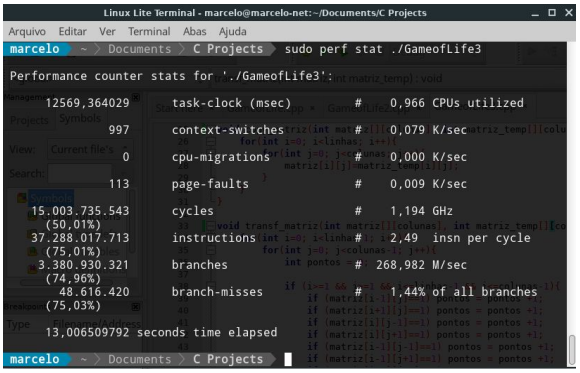
O segundo teste realizado foi precisão de previsão de ramificação.



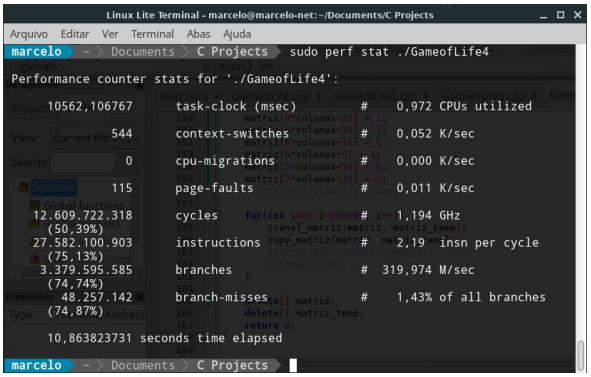
Game of Life:



Game of Life 2:



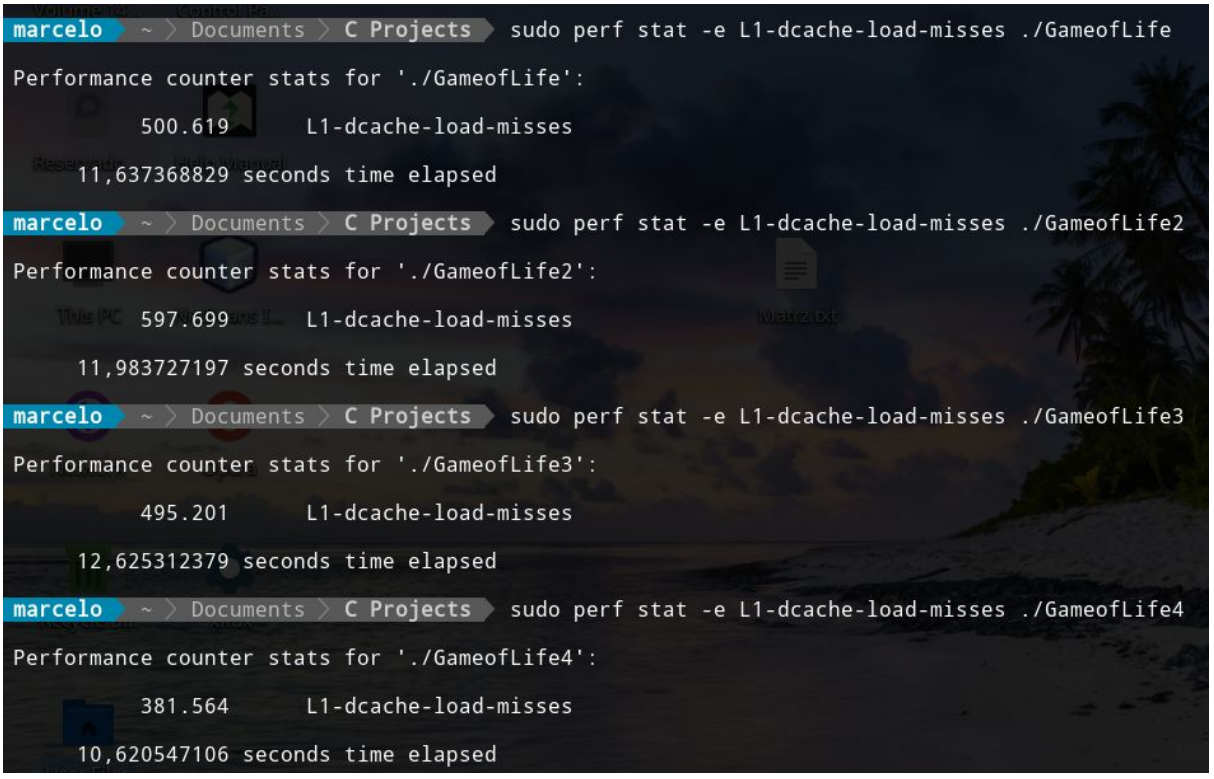
Game of Life 3:



Game of Life 4:

Como pode ser observado nas imagens acima a versão que utiliza apenas a alocação dinâmica das matrizes foi a que apresentou menos erros. O pior resultado foi a versão onde a busca nas matrizes é feita por coluna. Isso implica que a forma como a matriz é alocada na memória ou na pilha não reduz tanto o conflito de controle em processadores de arquitetura Pipeline, mas a forma como uma busca na matriz é realizada afeta sim o desempenho nesse caso. A predição de ramificação deixa de ser tão eficiente conforme a lógica da busca é alterada para colunas.

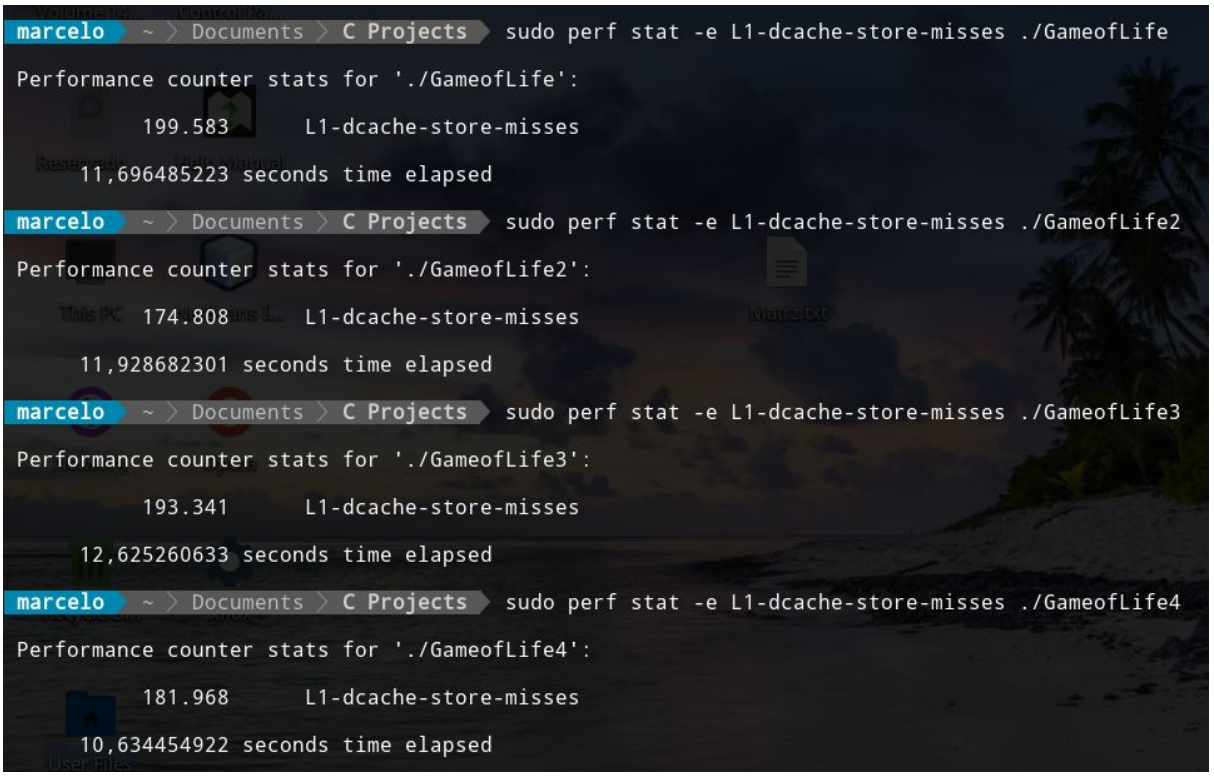
O terceiro teste foi o de falha leitura na cache (cache load miss)



Conforme pode ser observado a forma de alocação da matriz dinâmica ou local não interfere nesse teste. No entanto a versão com a representação lógica das matrizes em um vetor tem em torno de 20% menos erros de leitura na cache do que versão inicial com matrizes bidimensionais. Já as buscas realizadas em colunas causou um aumento de 20% nos erros de leitura na cache.

Isso vem do fato que a cache trabalha com localidade alocando as células que vem em seguida horizontalmente. A busca por colunas cria “misses” por não seguir essa localidade. Já a versão onde as matrizes são representadas em vetores segue melhor a lógica de alocação da cache e por isso tem menos erros. Isso interfere diretamente na velocidade da conclusão da tarefa.

O quarto teste foi o de falha escrita na cache (cache store miss)



Esse teste apresentou um estranho resultado. A busca por coluna nas matrizes da simulação mostrou menos “misses” de escrita na cache que a tradicional por linha. Esse evento pode ser devido a alguma especificação da simulação que interfere no processo de escrita na cache, mas não há como saber exatamente o porquê sem uma melhor avaliação dos processos do programa e do funcionamento das políticas de escrita na cache (write-through or write-back).

Hardware utilizado

Sistema Operacional: Linux Ubuntu 18.04.2 LTS (Linux Lite 4.2).
Códigos das versões da simulação em anexo.