Solving Classical Aproach of RNA Secondary Structure in Graphics Processing Unit

Marcos Negreiros
Mestrado Profissional em Computacão Aplicada, MPCOMP
Universidade Estadual do Ceara, UECE
Fortaleza, Brazil
negreiro@graphvs.com.br

Pedro Jorge de Abreu Figueredo
Laboratório de Computação Científica, LCC
Universidade Estadual do Ceara, UECE
Fortaleza, Brazil
pedro.jorge@aluno.uece.br

***Resumo*—Este artigo tem como intenção mostrar a utilização de um ambiente massivamente paralelo em GPU com a utilização da API CUDA da NVIDIA para abordar problemas que usam paradigma de Programação Dinâmica para serem resolvidos. Para o mesmo é utilizado o Problema Clássico da predição da RNA Secondary Structure mostrando a classificação do problema, a definição do modelo, criação da estrutura de memória a ser mantida em GPU com ênfase no coalesced access, sincronização de fases com sincronização interblocos e utilização de warp reduction. Gerando resultados com speedup de até 100 vezes em relação a implementação serial e mostrando a integração com a plataforma Gephi para melhor visualização da estrutura resultante.**

***Keywords*-Programação Dinamica; GPGPU; RNA; Warp Reduce; Sincronização InterBloco;**

I. Introduction

Algoritmos que usam programação dinâmica(DP) são geralmente algoritmos que resolvem problemas de optimização discretos, aonde deve se decompor um problema em problemas menores e ao resolver os mesmos pode-se utilizar uma função de composição para resolver o problema pai, essa função é a função recursiva do problema. Como nesses problemas o mesmo subproblema pode ser utilizado para resolver multiplos subproblemas em uma camada superior podemos memorizar a solução em uma tabela de "memoization"assim evitando ter que descer a partir folha na árvore de recursão novamente, se aplicarmos esse pensamento para toda a estrutura teremos um algoritmo que utiliza o paradigma DP. Porém, mesmo utilizando técnicas de DP multiplos desses algoritmos possuem complexidades temporais de valor elevado em CPU como o problema do RNA Secondary Structure Problem, Traveling Salesman Problem, Knapsack Problem como formulados em [1]. O que é proposto é seja feita a resolução de subproblemas em paralelo porém pra isso é necessario descobrir o nivel de independência entre os subproblemas e para isso temos a classificação de [2] das formulações DP:

- Serial — Se subproblemas dependem apenas de subproblemas de camadas anteriores.
- Non-serial — Se subproblemas não dependem apenas de subproblemas de camadas anteriores.
- Monodiac — Se a função recursiva possui apenas um fator de resolução.
- Polyadic — Se a função recursiva possui mais de um fator de resolução.

O algoritmo clássico de RNA Folding to find the Secondary Structure utiliza como critério criar o maior números de ligações entre bases seguindo o complemento de Watson-Crick, ou seja, dada as bases b_i e b_j $\in \{(A)denine, (C)ytosine, (G)uanine, (U)racil\}$ apenas pareamentos C-G e A-U são permitidos, devem haver no mínimo 4 bases entre i e j e as ligações de dois pares diferentes não podem se cruzar. Dados esse parâmetros e que $OPT(i, j)$ é o número máximo de bases entre i e j podemos chegar a seguinte função de recursão:

TODO:SUBFIGURE FUNÇÃO RECURSIVA a). TODO:SUBFIGURE dependencia triangular b).

Vendo a função a) podemos classifica-lá como Non-serial Polyadic Dynamic Programming(NPDP) esse tipo de DP se caracteriza por possuir um dependência triangular como mostrado em b) diferente das outras classificações que geralmente são retangulares. Trabalhos resolvendo RNA Folding em GPU podem ser encontrados em [3], [4] porém os mesmos usam critérios diferentes recorrência de zuker e nussinov, respectivamente. Um trabalho mais generalizado para resolução de NPDP pode ser encontrado [5], como problemas NPDP possuem um nível de dependência de dados dinâmica o trabalhos citados anteriormente precisam usar múltiplas chamadas de kernel perdendo certo nível de paralelismo, nesse é utilizado um método adaptativo de threads para mapear a NPDP do problema OMP(Optimal Matrix Parenthesization) em conjunto com método de sincronização inter-blocos [6] agora usando uma chamada de kernel e utilizando de forma mais eficiente a memória compartilhada, mas mesmo que métodos NPDP possuam caracteristicas semelhantes as estruturas de acessos usadas são diferentes para cada tipo de problema. O método proposto nesse artigo utiliza-se de método de hashing para mapear a estrutura de memoization em mémoria global, acessa-lá de forma coalesced e fixando o número de threads, a ultilização de warp reduction minimizando utilização de mémoria com-

partilhada, sincronização inter-blocos permitindo evitando sincronização via CPU permitindo speed ups de até 100 vezes contra CPU.

II. CUDA API

III. The Parallel Model

Modelos NPDP como o problema em estudo possuem subproblemas independentes entre si na mesma diagonal assim se considerarmos o passo de resolver paralelamente uma diagonal como uma fase depois de k fases teremos a solução veja figura c).

TODO:SUBFIGURE c) matrix sendo resolvida em onda

Além de resolvermos o problema seguindo um modelo em onda devemos resolver cada subproblema em paralelo pois os mesmos escolhem um array de subproblemas e escolhem o maior valor e isto pode ser solucionado fazendo uma redução como em d).

TODO:SUBFIGURE d) subproblema gera array reduzir array.

Dadas as considerações anteriores faremos o seguinte, cada subproblema será resolvido em um SM e todos os SM's tem que estar na mesma fase de execução ao resolverem uma diagonal os SM's podem passar para próxima fase para garantir isso usaremos o métodos de sincronização inter-blocos chamado gpu_ sync, cada subproblema sendo resolvido irá escolher os elementos que seram resolvidos pelo método de "warp reduce"com um bloco de tamanho fixo no caso 1024 threads e ao final da fase k teremos a solução na matrix de memoization em memória global no device.

TODO:algorithmcs e) algoritmo resolução

IV. Sincronização Inter-bloco

Em [6] são mostrados dois métodos de sincronização um lock-based e outro lock-free por motivos de performance foi ultilizado o método lock-free. Nesse método possuimos um array de entrada, um array de saída e um valor objetivo que será diferente para cada fase, na medida com que as threads chamam gpu_sync (figura f)) as threads entram na barreira __syncthreads(), a thread 0 em cada bloco muda o valor do seu respectivo bloco no array de entrada para o valor objetivo e fica em espera verificando se o seu respectivo valor no array de saida é igual ao valor objetivo, ao perceber a igualdade a thread irá entrar na barreira e permitira que o bloco continue a execução para próxima fase, o bloco número 1 é responsável por verificar se todo os valores do array de entrada são iguais ao objetivo então irá preencher o array de saída com valor objetivo.

TODO:Subfigure f) algoritmo

V. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

The authors would like to thank... more thanks here

Referências

[1] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[2] V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[3] R. D. Chamberlain, J. D. Buhler, and A. C. Jacob, "Rapid rna folding: Analysis and acceleration of the zuker recurrence," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 00, pp. 87–94, 2010.

[4] S. S. Junjie Li, Sanjay Ranka, "Multicore and gpu algorithms for nussinov rna folding," *2013 IEEE 3rd International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*, vol. 00, pp. 1–2, 2013.

[5] W. chun Feng, C.-C. Wu, J.-Y. Ke, and H. Lin, "Optimizing dynamic programming on graphics processing units via adaptive thread-level parallelism," *Parallel and Distributed Systems, International Conference on*, vol. 00, pp. 96–103, 2011.

[6] W. chun Feng and S. Xiao, "Inter-block gpu communication via fast barrier synchronization," *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, vol. 00, pp. 1–12, 2010.