



**Ricardo Alexandre do Rosário Ribeiro**

Master of Science

## **Protein docking GPU acceleration**

Dissertation plan submitted in partial fulfillment  
of the requirements for the degree of

Master of Science in  
**Computer Science and Informatics Engineering**

Adviser: Ludwig Krippahl, Full Professor,  
NOVA University of Lisbon

Co-adviser: Hervé Paulino, Associate  
Professor, NOVA University of Lisbon



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

July, 2018



## RESUMO

---

Na área científica da bio informática, o estudo das interações entre as proteínas tem sido objeto de interesse. No entanto conseguir determinar computacionalmente e com precisão como é que as mesmas se unem é difícil.

O presente documento aborda uma proposição para a paralelização do algoritmo BiGGER implementado pelo prof. Ludwig Krippahl e outros professores associados à sua criação. A implementação será feita recorrendo a técnicas de computação acelerada i.e. utilizar o GPU da máquina em que corre o algoritmo para auxiliar o CPU na computação que é necessária.

Tendo mais recursos à disposição, é esperado que o tempo de execução do BiGGER baixe drasticamente por consequencia do aumento significativo de performance face à versão sequencial. Em caso de sucesso, a complexidade futura do algoritmo permitirá a adição de mais vantagens face aos seus concorrentes. Por consequencia deste aumento de performance, uma proposta de valor para quem pretenda utilizar o open-chemera como ferramenta de trabalho eficiente no estudo das interações entre as proteínas em qualquer máquina que tenha uma placa gráfica com as características adequadas.

**Palavras-chave:** proteínas, docagem de proteínas, computação acelerada, GPU, Bio-Informática, BiGGER

---



## ABSTRACT

---

The dissertation must contain two versions of the abstract, one in the same language as the main text, another in a different language. The package assumes that the two languages under consideration are always Portuguese and English.

The package will sort the abstracts in the appropriate order. This means that the first abstract will be in the same language as the main text, followed by the abstract in the other language, and then followed by the main text. For example, if the dissertation is written in Portuguese, first will come the summary in Portuguese and then in English, followed by the main text in Portuguese. If the dissertation is written in English, first will come the summary in English and then in Portuguese, followed by the main text in English.

The abstract should not exceed one page and should answer the following questions:

- What's the problem?
- Why is it interesting?
- What's the solution?
- What follows from the solution?

**Keywords:** Keywords (in English) ...

---



# ÍNDICE

<b>Lista de Figuras</b>	<b>ix</b>
<b>Lista de Tabelas</b>	<b>xi</b>
<b>Listagens</b>	<b>xiii</b>
<b>Glossário</b>	<b>xv</b>
<b>Siglas</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Estado-da-arte</b>	<b>3</b>
2.1 Modelos de docagem em relação à rigidez da superfície . . . . .	3
2.1.1 Docagem Flexível . . . . .	3
2.1.2 Docagem Semi-Flexível . . . . .	3
2.1.3 Docagem Rígida . . . . .	4
2.2 Algoritmo Geometric Hashing . . . . .	4
2.3 Fast Fourier Transform . . . . .	4
2.4 ZDOCK . . . . .	5
2.5 O algoritmo BiGGER para docagem de proteínas . . . . .	5
2.6 BoGIE . . . . .	6
2.7 O uso de GPU como ferramenta auxiliar . . . . .	7
2.7.1 APIs para computação acelerada . . . . .	8
2.8 Softwares com otimizações ao FFT usando GPU . . . . .	9
2.8.1 Megadock . . . . .	9
<b>3 Plano de trabalhos para a Elaboração da dissertação</b>	<b>11</b>
3.1 Possibilidades de otimização . . . . .	11
3.2 Desafios . . . . .	12
3.3 Plano de Trabalhos . . . . .	13
<b>Bibliografia</b>	<b>15</b>





## LISTA DE FIGURAS

2.1	Representação em 2D das matrizes resultantes do segundo passo do BoGIE, as células preenchidas a tracejado diagonal correspondem à matriz de superfície, as com pontos correspondem à matriz <i>core</i> e a núvem com tracejado contínuo representa o corte associado à esfera de van der waals com a proteína localizada ao centro [9]. . . . .	7
2.2	Esquema ilustrativo da diferença entre o hardware do CPU e do GPU, demonstrando também a diferença de cores entre os 2. Imagem da NVIDIA . . . . .	8
3.1	Representação gráfica do ID geral de uma thread num array dividido por blocos, em CUDA. Adaptações de [11] [13] . . . . .	12



## LISTA DE TABELAS

2.1	Tempos de cálculo das versões iniciais do Megadock em relação ao ZDOCK [8].	10
-----	---	----



## LISTAGENS



## GLOSSÁRIO

aliquam	tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris..
computer	An electronic device which is capable of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information or signals..
cras viverra	metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat..
donec nonummy	pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo..
integer sapien	est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus..
lorem ipsum	dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris..
maecenas lacinia	nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem..
morbi ac	orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus..
morbi dolor	nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum..

nam lacus	libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi..
nam dui	ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo..
name arcu	libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo..
nulla malesuada	porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis..
sed lacinia	nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus..



## SIGLAS

aaa	acornym aaa.
aab	acornym aab.
aba	acornym aba.
abbrev	abbreviation of a longer text.
AEU	adipiscing elit ut.
AFM	aenean faucibus morbi.
AMD	a magna donec.
ANP	ac nunc praesent.
ATG	amet tortor gravida.
AVF	adipiscing vitae felis.
bbb	acornym bbb.
CAS	curabitur auctor semper.
CDG	curabitur dictum gravida.
CEA	congue eu accumsan.
CIV	consectetuer id vulputate.
DIA	duis eget orci.
DNM	dolor nulla malesuada.
DNMC	duis nibh mi congue.
DRN	dignissim rutrum nam.
EII	est iaculis in.
ENE	et netus et.
EPA	eu pulvinar at.
ESQ	eleifend sagittis quis.
ESV	eget sem vel.
ETS	eu tellus sit.

FUP    fringilla ultrices phasellus.

LID    lorem ipsum dolor.

LNE    libero nonummy eget.

LUB    leo ultrices bibendum.

LVU    lectus vestibulum urna.

MAC    mollis ac nulla.

MFA    malesuada fames ac.

MNA    mauris nam arcu.

MTS    morbi tristique senectus.

NDV    nulla donec varius.

NPH    neque pellentesque habitant.

OER    orci eget risus.

PEV    purus elit vestibulum.

PIS    placerat integer sapien.

PQV    pretium quis viverra.

SAO    sit amet orci.

SNE    sem nulla et.

STC    sit amet consectetur.

TEM    turpis egestas mauris.

ULC    ut leo cras.

UPA    ut placerat ac.

VAE    vehicula augue eu.

VMR    viverra metus rhoncus.

xpto    and extension of a xpto xpto xpto xpto xpto xpto xpto xpto xpto xpto  
xpto xpto xpto xpto xpto xpto xpto xpto.





## INTRODUCTION

*This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/>.*



## ESTADO-DA-ARTE

### 2.1 Modelos de docagem em relação à rigidez da superfície

Com o passar do tempo cada vez mais algoritmos e respetivas adaptações para simular a docagem dos complexos de proteínas têm surgido, adoptando modelos que formulam hipóteses em relação às características dos elementos envolventes.

Um algoritmo pode ser classificado em função da forma que trata a rigidez da superfície dos pares de proteínas a juntar ou até mesmo pelo modelo matemático que os algoritmos seguem, como por exemplo se aplicam a Fast Fourier Transform ou não.

Serão abordadas nas próximas subsecções 3 modelos de docagem: flexível, semi-flexível e rígida.

O modelo semi-flexível, no entanto, não será tão considerado, já que a maior parte dos artigos publicados assim como algoritmos seguem os dois modelos anteriores.

#### 2.1.1 Docagem Flexível

Em que ambos os complexos receptor e ligando são considerados como sendo corpos flexíveis e adaptáveis sendo, no entanto, a mesma flexibilidade interpretada pelo algoritmo de forma simplificada ou limitada, e por consequência pode-se aplicar um modelo através de simulações de docagem.

#### 2.1.2 Docagem Semi-Flexível

Um dos corpos é considerado rígido e o outro não, em situações normais este tipo de algoritmos trata o ligando como se fosse a proteína flexível, já que esta é mais pequena que a sua parceira, tendo assim uma maior probabilidade de mudar a sua forma, outra justificação tem a ver com os custos de computação serem mais baixos do que se considerarmos os receptores como flexíveis.

### 2.1.3 Docagem Rígida

O par é considerado como sendo rígido na sua integridade, sendo também considerado que na docagem entre os dois corpos uma das proteínas irá acabar por penetrar a outra o que leva a que se tenha de adaptar o conjunto de soluções para o problema em seis dimensões, 3 para a rotação e 3 para a translação.

Apesar de se considerarem as superfícies de ambos como rígidos, considera-se que ocorrem variações na superfície permitindo que haja penetrações inter-moleculares.

Este modelo tem a vantagem a ser o mais rápido dos 3 a considerar, sendo capaz de explorar as superfícies de ambos os elementos e traduzir para uma base de dados de docagem [3].

## 2.2 Algoritmo Geometric Hashing

A desenvolver amanhã...

## 2.3 Fast Fourier Transform

Algoritmo baseado em previsões geométricas sobre o complexo formado após a interação entre as proteínas, representando o método a partir do qual ocorreram adaptações que levaram origem à criação do BiGGER [9]. É utilizado em muitos softwares específicos para *protein docking* ( eg. Megadock 4.0, Autodock,... ).

A origem deste método remonta ao artigo de Katchalski Katzir et al [4] onde se considera que ambos os pares são corpos rígidos, o que no âmbito do ponto 2.1 pode-se assumir como docagem rígida.

À semelhança do BiGGER, este método também recorre a uma estimativa da complementaridade entre as superfícies dos pares que compõem a PPI, apenas existe como requisito o conhecimento da estrutura tri-dimensional destes.

A metodologia com que são determinadas as possibilidades consiste, de forma resumida, nos passos:

1. Considerando duas moléculas  $a$  e  $b$ , correspondentes ao receptor e ao ligando, é determinada uma projeção das coordenadas atómicas de ambos para uma grelha tri-dimensional ( $N^3$ ).

São também determinadas duas funções discretas sobre as coordenadas, em que cada ponto da grelha assume o valor 0 ou 1, se a coordenada corresponde a uma posição interna à molécula e 0 caso contrário.

O limite para atribuição dos valores está definido por um raio  $r$  associado a uma nuvem de van der waals, que assume uma curvatura semelhante à que se pode determinar na figura 2.2.



2. Determinação da região de fronteira, em que a função discreta determinada no ponto anterior é estendida para suportar os pontos fronteiros: 1 neste passo é atribuído a coordenadas atómicas localizadas na fronteira, uma variável associada a coordenadas internas e 0 a externas. A esta função é designada função discreta de Fourier (DFT).
3. É determinada a função de correlação associada às orientações entre as duas moléculas, sendo considerado que a molécula *a* está fixa enquanto *b* pode ter orientações variadas. Sobre um eixo *xyz* os ângulos que a orientação do ligando pode formar variam entre  $360 \times 360 \times 180 \Delta^3$ .

Em termos de complexidade temporal, executar estes passos assume uma ordem de complexidade  $O(N^3 * \log_2(N^3))$ [5].

Considerou-se fazer um estudo deste método importante pois tal como foi explicitado no princípio desta secção foi a partir do trabalho de Katchalski Katzir e colegas sobre o FFT que foi estudada a abordagem para o BiGGER[5]. Tendo em conta que os passos aqui descritos e as fases do bigger descritas na secção 2.5 são muito semelhantes, para se perceber como paralelizar o BiGGER é necessário entender como este funciona, e por consequência, como funciona o FFT.

Outra razão para se ter feito um estudo sobre esta técnica aborda poder justificar onde é que o BiGGER é mais forte do que o FFT aquando na descrição de resultados obtidos na fase da elaboração.

É esperado que o Open-chemera com o BiGGER paralelizado assuma uma performance superior a qualquer um dos casos estudados, que utiliza cuFFT .

## 2.4 ZDOCK

### 2.5 O algoritmo BiGGER para docagem de proteínas

Este algoritmo (acrónimo para *Bimolecular complex Generation with Global Evaluation and Ranking*) [5] de acordo com a classificação referida no ponto 2.1 enquadra-se na doutrina de docagem rígida[9].

Permite resolver o problema de conseguir prever o ajuntamento dos complexos proteicos, consistindo essencialmente em dois passos: o primeiro efectua uma redução de possíveis configurações resultantes de passos de translação e rotação numa magnitude de cerca de  $10^{15}$  configurações para poucos milhares das anteriores, através do algoritmo BoGIE relatado no ponto 2.6.

A segunda fase do algoritmo consiste em aplicar metodologias de aprendizagem automática de modo a que se possa prever qual das configurações resultantes corresponde ao melhor ajuste entre os dois complexos, isto é, a que tem o score mais elevado.

Em termos de complexidade temporal, este algoritmo assume valores mais optimais ( $O(N^{2,8})$ [9]) do que os algoritmos que recorrem ao Fast Fourier Transform.

O motivo pelo qual das duas espécies de algoritmos, o BiGGER assume-se com performance superior em termos de computação, deve-se ao facto de este ter sido implementado com diversas optimizações face aos algoritmos FFT, uma delas consiste no uso de uma heurística mais eficiente no passo da eliminação de possibilidades: descarta situações em que existem sobreposições entre cores ou até mesmo situações que não cumprem com os limites impostos nas restrições introduzidas [9].

O tempo de execução do algoritmo, segundo os autores do algoritmo estava situado entre as 2H e as 8H, dependendo do par de proteínas, em contraste com o tempo de execução para FTT que ronda as 6H, com um CPU do ano de 2000 (Intel Pentium II 450 MHz dispõe apenas 1 core)[9].

Segundo a lei de Moore, o número de transistors presentes num CPU duplica a cada 2 anos, e por consequência a capacidade computacional, pelo que num computador em 2018 o tempo de execução do BiGGER provavelmente será menor, demorando entre 1H e 4H por exemplo.

## 2.6 BoGIE

Acrónimo para *Boolean Geometric Interaction Evaluation*[5] [9], é um algoritmo de pesquisa em grelha utilizado na primeira fase do BiGGER, que é referido no ponto seguinte, mais precisamente na amostragem da população total de configurações possíveis para milhares.

Existem dois processos principais a considerar, sendo o primeiro a definição de uma matriz tridimensional de booleanos em que cada posição da matriz representa uma parcela da forma que o complexo assume.

Um nó da matriz assume valor 1, se a célula corresponde a uma parcela da proteína cujo centro se encontra a uma distancia tri-dimensional, designada por esfera de Van der Waals, de qualquer outro átomo pertencente a outra proteína, e o valor 0, se a mesma corresponder a frações do complexo que são consideradas como externas.

O segundo passo gera duas matrizes de valores booleanos semelhantes às anteriores para cada um dos elementos do par: a matriz de superfície (*surface matrix*) e a matriz central (*core matrix*) tal como está ilustrado graficamente na figura 2.2.

Os elementos celulares que ocupam a matriz de superfície são aqueles que na matriz inicial do passo anterior assumiram valor 1 mas tinham vizinhos com valor 0, ou seja, pretende-se os pontos de fronteira.

Na segunda matriz constam as células em que quer o seu valor, quer o das suas células vizinhas assumem valor verdadeiro, o que corresponde a posições em que o seu centro está próximo do centro do complexo proteico ou podendo até mesmo coincidir.

A forma de garantir que se consegue obter a superfície molecular da proteína é através da operação lógica XOR (OU exclusivo), que terá como output 1 apenas nos pontos da fronteira, pois é aqui que o resultado do XOR associado aos dois pontos seleccionados dá

o valor verdadeiro, já que os valores entre as duas células são diferentes e falso se forem iguais.

Sendo assim a complexidade deste algoritmo está associada mais com o primeiro passo do que com o segundo, já que este ultimo depende do output da matriz resultante do primeiro passo e apenas executa um conjunto de operações XOR o que não é assim tão custoso em termos de memória e tempo comparando com a medição para cada célula de uma distância.

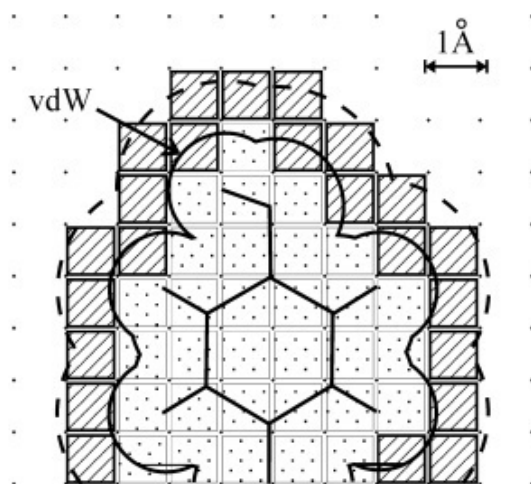


Figura 2.1: Representação em 2D das matrizes resultantes do segundo passo do BoGIE, as células preenchidas a tracejado diagonal correspondem à matriz de superfície, as com pontos correspondem à matriz *core* e a nuvem com tracejado contínuo representa o corte associado à esfera de van der waals com a proteína localizada ao centro [9].

De notar, no entanto, que ambos os passos podem ser otimizados recorrendo ao GPU, no capítulo 3 serão detalhadas possíveis abordagens à paralelização desta etapa do BiGGER, podendo trazer melhorias para além do uso do XOR.

## 2.7 O uso de GPU como ferramenta auxiliar

GPU( *Graphical Processor Unit*) consiste na unidade de processamento gráfico existente na placa gráfica instalada em qualquer computador, sendo especializada em processamento gráfico, mais precisamente renderização de gráficos 3D.

No entanto o GPU também é adequado para processamentos alternativos aos jogos, que são igualmente intensivos demais para o CPU. Exemplos de aplicações podem variar de calculo financeiro até aplicações bio-informáticas, como é o caso do protein-docking.

O GPU consegue ser mais poderoso a executar instruções com custos de complexidade temporal elevado do que necessariamente o CPU, que tem um número de cores limitado. [6]

Um programador que pretenda implementar uma paralelização para a versão sequencial de um dado programa, recorrendo ao CPU, necessita de conhecimentos de programação com *threads* assim como gerir o acesso exclusivo a variáveis partilhadas entre as mesmas, o que pode ser feito por gestão de locks ou por instruções atômicas.

O resultado consiste num programa paralelizado mas com um speedup limitado comparando com uma solução que use aceleradores no GPU.

O GPU acaba por ser uma ferramenta mais potente para paralelizar programas já que a sua arquitetura é consistente em uma quantidade variável de multi-processadores (*streaming multi-processors*) [10] que por si só têm um conjunto de processadores escalares (SPs) que são também conhecidos como os cores do GPU, pode ter até 512 cores no total, se a arquitetura for da espécie Fermi. [12]

Em termos de arquitetura está também presente uma zona de memória global que pode variar entre MBs e GBs, e pode ser partilhada entre os diversos SPs.

Porém continuamos a necessitar do CPU para comunicar com o GPU, em que este último funciona como *device* e o CPU funciona como *host* numa interface de comunicação de dados entre os 2. Sendo assim é essencial usar o GPU como utensílio na paralelização de algoritmos que façam computações muito pesadas, como é o caso das etapas presentes nos algoritmos destacados nas secções anteriores.

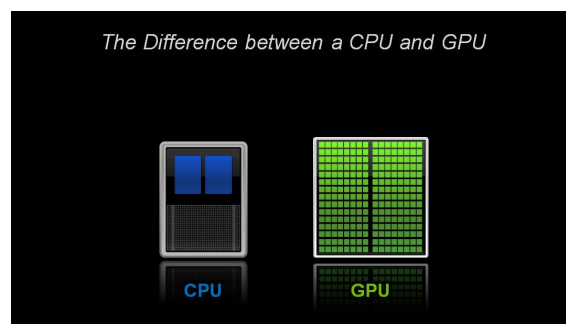


Figura 2.2: Esquema ilustrativo da diferença entre o hardware do CPU e do GPU, demonstrando também a diferença de cores entre os 2. Imagem da NVIDIA

### 2.7.1 APIs para computação acelerada

Em termos de programação em GPUs existe como API o CUDA (*Compute Unified Device Architecture*) que foi implementado pela NVIDIA

O CPU executa a parte sequencial do programa enquanto que o GPU, com a sua quantidade numerosa de cores, executa a parte que requer computação mais intensiva como por exemplo multiplicação de matrizes (*matmult*), através de invocações especiais que se denominam *kernels*. [6]

De notar que é necessário fazer previamente as alocações respetivas de memória do host para o device, que são feitas por parte do CPU.

Um programador que queira utilizar CUDA para paralelizar o seu programa, deve transferir a toolkit respetiva ao sistema operativo em que pretende realizar o projeto, esta toolkit está disponível na página da NVIDIA.

No entanto também existem mais APIs/frameworks para o efeito, como por OpenCL(*Open computing language*) que é adequado para programação paralela em Free pascal enquanto que o CUDA é mais utilizado para C/C++ ou até mesmo Python (pyCUDA).

Ambos são semelhantes no sentido de implementar o protocolo de comunicações entre o CPU e o GPU, e de permitir invocar kernels para este último computar a parte intensiva.

## 2.8 Softwares com optimizações ao FFT usando GPU

Já existem vários softwares e artigos dedicados à introdução de paralelizações ao FFT [10] o que muitos têm em comum é o facto de utilizarem CUDA, neste caso utilizam CuFFT.

O CuFFT é uma versão própria do CUDA que fornece as implementações necessárias para que a performance do algoritmo seja aumentada em 10x, face às versões que utilizam o CPU para o processamento respetivo[1].

### 2.8.1 Megadock

Um exemplo desta espécie de softwares específicos para docagem é o Megadock 4.0, este software é de origem japonesa, que melhora em relação ao Megadock 1.0 no sentido de ter implementações no âmbito da computação acelerada [8].

Este programa é adequado para máquinas que têm muitas cores de GPU e CPU à disposição, características típicas de supercomputadores, não sendo, por observação própria, adequado para utilização em computadores pessoais de forma a obter os resultados óptimos.

Foi implementado para GPU não só por CUDA como também por MPI (Message Passing Interface) e OpenMP.

O MPI, tal como a sigla especifica, é uma API de interface de comunicação de mensagens entre processos a correr nos nós de um cluster[2]

O Megadock 4.0 é um enhancement em relação à versão 1.1 no sentido de suportar o modelo master-worker.

Este modelo é uma instância do paradigma SPMD (Single Program Multiple Data).

O funcionamento do Megadock 4.0 envolve a criação de um processo master que faz a aquisição de uma lista de pares de proteínas e cria jobs incidentes sobre a lista para os workers.

Estes, por sua vez, distribuem o trabalho de calcular a rotação do ligando em cada nó da lista, pelos diversos GPUs e CPUs do nó do cluster. A distribuição pelos GPUs de cada nó é feita por CUDA e pelos CPUs por OpenMP [7].

	Megadock 1.0	Megadock 2.0	Megadock 2.1	ZDOCK 3.0
Tempo de cálculo (min)	13.3	14.7	16.6	124.6

Tabela 2.1: Tempos de cálculo das versões iniciais do Megadock em relação ao ZDOCK [8].

Uma das vantagens que este protocolo assume é a tolerancia de falhas pois o nó master consegue supervisionar os resultados dos jobs executados pelos workers, além disso é escalável com o número de elementos que compõem o cluster.

No decorrer da elaboração do projeto da dissertação, se a abordagem for implementada de forma a que se possam efectuar testes de performance, serão feitas comparações com os resultados teóricos dos artigos dos autores do Megadock ( tabela 2.1).

Os resultados referidos foram obtidos por execução no supercomputador disponível no Instituto de Tecnologia de Tóquio, TSUBAME 2.0. A dimensão amostral consiste no número total de combinações de estruturas receptoras (1936) e no mesmo número para estruturas ligandas (14400) . No caso do Megadock 4.0, o valor a considerar para o tempo de cálculo ronda as 5.71h para meio milhão de jobs e 11.51H para 1 milhão de jobs. Para este último teste foi utilizado a versão 2.3 do TSUBAME[7].

*Pending detalhar a implementação, introduzir mais casos*

## PLANO DE TRABALHOS PARA A ELABORAÇÃO DA DISSERTAÇÃO

### 3.1 Possibilidades de otimização

O programa que serve de interface gráfica ao BiGGER, que se pode fazer a clonagem do repositório em <https://github.com/lkrippahl/Open-Chemera> encontra-se implementado em Free Pascal, 97.6% do código total, ao acordo com o que foi abordado previamente, o trabalho a realizar na elaboração incide sobre o package **docking** mais precisamente às unidades **bogie.pas** e **dockdomains.pas**. O **bogie.pas** consiste no módulo de docagem geométrica e na unidade **dockdomains** são determinados os domínios nos três eixos para a simulação da docagem geométrica.

O trabalho poderá abarcar a paralelização de mais unidades presentes no pacote o que só garante melhorias adicionais à performance do Open Chemera, por exemplo na secção 3.2 é abordada uma paralelização à unidade **linegrids.pas**, que é a unidade onde é feito o cálculo das regiões de superfície e core dos pares assim como a determinação das grelhas para a superfície 3D dos mesmos, para o efeito de complementar a resolução do desafio estipulado na secção. As principais alterações serão, no entanto, focadas nos dois referidos anteriormente.

Face à possibilidade de não existir nenhuma versão do CUDA para programar paralelizações em Free Pascal.

Põe-se de lado esta última alternativa em troca de uma outra que recorra à framework OpenCL, que é suportada pelo Lazarus (IDE a utilizar durante a fase de elaboração do tema).

Poderão vir a ser implementados para a paralelização das duas unidades, kernels que executam operações de mapeamento, em que o espaço de possibilidades a tratar na primeira fase do BiGGER poderá ser associado a uma estrutura de dados (figura 3.1),

dividida por blocos em que estes últimos serão constituídos por casas indexadas pelo ID da thread correspondente. Pelo que a indexação geral estará associada a uma fórmula que envolva a posição da thread no bloco e o número de bloco. Para cada uma das posições o core do GPU executará a função que determina se a configuração é aceite ou não.

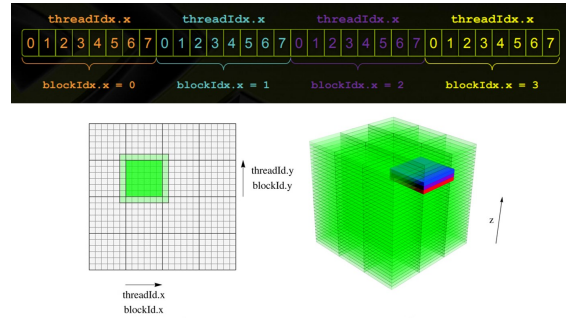


Figura 3.1: Representação gráfica do ID geral de uma thread num array dividido por blocos, em CUDA. Adaptações de [11] [13]

Este esquema de indexação de threads também existe em OpenCL, por invocações próprias na sua sintaxe. *Pending, ponderar mais tipos de kernels que possam ser implementados, para fazer operações de redução/scan/stencil*

## 3.2 Desafios

Os desafios incidem-se sobre dois pontos:

- **Criação das grelhas tri-dimensionais** : Tal como foi elaborado no capítulo anterior, o passo inicial do BiGGER consiste na criação de grelhas tri-dimensionais, de booleanos, que assumem valor 1 ou 0 se a posição respetiva na grelha corresponde a uma posição atómica da proteína.

Analisando o código que se pode encontrar na unidade dockdomains.pas pode-se verificar que existem uma certa diversidade nas possibilidades de optimização.

O código presente nesta unidade contém um conjunto de *procedures* que executam cadeias de ciclos for/while, em teoria é possível paralelizar o dockdomains recorrendo a kernels que executam operações de map, de forma a reduzir a carga de computação.

Paralelizações adicionais poderão ser feitas na unit linegrids.pas que permitem trazer melhorias extra de performance, no entanto serão alterações complementares, segundo a documentação inicial do código fonte da unidade, os segmentos gerados por esta unidade são referenciados apenas ao eixo z, sendo então uma unidade auxiliar para o BiGGER para a indexação da matriz tri-dimensional por parte deste.

- **Pesquisa de sobreposições** :



### **3.3 Plano de Trabalhos**



## BIBLIOGRAFIA

- [1] *cuFFT*. 2018. URL: <https://developer.nvidia.com/cufft>.
- [2] W. D. Gropp, W. Gropp, E. Lusk e A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.
- [3] H. Inbal, M. Buyong, W. Haim e N. Ruth. “Principles of docking: An overview of search algorithms and a guide to scoring functions”. Em: *Proteins: Structure, Function, and Bioinformatics* 47.4 (), pp. 409–443. DOI: 10.1002/prot.10115. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.10115>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.10115>.
- [4] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. A. Friesem, C. Aflalo e I. A. Vakser. “Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques.” Em: *Proceedings of the National Academy of Sciences* 89.6 (1992), 2195–2199. DOI: 10.1073/pnas.89.6.2195.
- [5] L. Krippahl. *Integrating protein structural information*. 2003.
- [6] C. Nvidia. “Nvidia cuda c programming guide”. Em: *Nvidia Corporation* 120.18 (2011), p. 8.
- [7] M. Ohue, T. Shimoda, S. Suzuki, Y. Matsuzaki, T. Ishida e Y. Akiyama. “MEGA-DOCK 4.0: an ultra-high-performance protein-protein docking software for heterogeneous supercomputers”. Em: *Bioinformatics* 30.22 (2014), pp. 3281–3283. DOI: 10.1093/bioinformatics/btu532. eprint: [/oup/backfile/content\\$public/journal/bioinformatics/30/22/10.1093\\_bioinformatics\\_btu532/2/btu532.pdf](http://oup/backfile/content$public/journal/bioinformatics/30/22/10.1093_bioinformatics_btu532/2/btu532.pdf). URL: <http://dx.doi.org/10.1093/bioinformatics/btu532>.
- [8] M. Ohue, Y. Matsuzaki, N. Uchikoga, T. Ishida e Y. Akiyama. “MEGADOCK: an all-to-all protein-protein interaction prediction system using tertiary structure data”. Em: *Protein and peptide letters* 21.8 (2014), pp. 766–778.
- [9] P. N. Palma, L. Krippahl, J. E. Wampler e J. Moura. “BIGGER: A new (soft) docking algorithm for predicting protein interactions”. Em: 39 (jun. de 2000), pp. 372–84.
- [10] Ritchie, D. W., Venkatraman e Vishwesh. *Ultra-fast FFT protein docking on graphics processors | Bioinformatics | Oxford Academic*. 2010. URL: <https://academic.oup.com/bioinformatics/article/26/19/2398/229220>.

## BIBLIOGRAFIA

---

- [11] J. Sainio. “CUDA EASY - a GPU accelerated cosmological lattice program”. Em: *Computer Physics Communications* 181 (2010), pp. 906–912.
- [12] C. M. Wittenbrink, E. Kilgariff e A. Prabhu. “Fermi GF100 GPU architecture”. Em: *IEEE Micro* 31.2 (2011), pp. 50–59.
- [13] C. Zeller. “Cuda c/c++ basics”. Em: *NVIDIA Corporation* (2011).