Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

UMA API DE COMUNICAÇÃO PARA ACELERAÇÃO POR HARDWARE DE SIMULADORES MOLECULARES

MAICON APARECIDO SARTIN

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre
Março de 2009

“Só existem dois dias no ano que nada pode ser feito. Um se chama ontem e o outro se chama amanhã, portanto hoje é o dia certo para amar, acreditar, fazer e principalmente viver.”

Dalai Lama

AGRADECIMENTOS

Agradeço primeiramente a Deus, e também a toda a minha família pelo exemplo de honestidade, força, determinação, profissionalismo, respeito e apoio incondicional. Gostaria de agradecer a compreensão e o carinho e acima de tudo a paciência e muito amor da minha esposa Eliana e da minha filha Yasmin que me apoiaram e motivaram, mesmo distantes, sempre me ofereceram suportes e acreditaram na finalização e conclusão desse trabalho. Ao meu cunhado Edimar pela solidariedade e apoio.

Aos amigos do MINTER que me ajudaram com suas opiniões, companheirismo durante a elaboração deste trabalho, em todo o mestrado e nos períodos de estadia em Porto Alegre.

Agradeço ao meu orientador, Ney Laert Vilar Calazans, uma pessoa que foi muito importante na minha jornada, pelo incentivo, apoio, atenção, dedicação e principalmente na paciência em todo o mestrado. Ao professor Osmar Norberto de Souza pelas informações, instruções e na dedicação nos esclarecimentos das dúvidas encontradas. Ao professor João Batista por informações e direcionamentos apresentados.

Aos colegas de trabalho do GAPH, em especial ao Adilson Mohr, Edson Moreno e Matheus Trevisan, que não mediram esforços e atenção nos momentos que precisei. Gostaria de agradecer o apoio financeiro da FAPEMAT e UNEMAT.

Muito Obrigado.

UMA API DE COMUNICAÇÃO PARA ACELERAÇÃO POR HARDWARE DE SIMULADORES MOLECULARES

RESUMO

A evolução da tecnologia de fabricação de circuitos integrados continua obedecendo à lei de Moore. Entretanto, aplicações científicas cada vez mais necessitam de recursos de alto desempenho computacional, motivando pesquisadores a propor a aceleração por hardware dedicado para aumentar o desempenho destas aplicações. Freqüentemente, devido à necessidade de rapidez no projeto de tais aplicações, empregam-se técnicas de projeto com emprego de hardware reconfigurável.

Atualmente, há um grande aumento em pesquisas de biofísica molecular com o objetivo principal na concepção de fármacos. Porém, para se chegar até a droga e a possível cura de alguma doença, diversos procedimentos devem ser empreendidos. Como exemplos podem ser citados experimentos para determinar o comportamento de moléculas simples ou de proteínas. As simulações por dinâmica molecular aportam uma variedade de informações do sistema molecular em questão. Entretanto, para se executar estas simulações é necessário o auxílio de recursos computacionais de alto desempenho, devido à elevada quantidade de cálculos a efetuar, à quantidade de informações geradas e à necessidade destas informações e resultados em períodos curtos de tempo, tornando a exigência por computação de alto desempenho uma característica básica desta área.

Para suprir a exigência computacional de simulações por dinâmica molecular existem plataformas baseadas em FPGAs, que são largamente utilizadas como aceleradores de hardware de aplicações com alto custo computacional. FPGAs são amplamente disponíveis e permitem realizar rapidamente o projeto e a implementação de hardware com alto desempenho se comparado a software executando em processadores de propósito geral.

A principal contribuição deste trabalho é uma proposta de método de comunicação entre uma máquina hospedeira e uma plataforma de hardware reconfigurável baseada em FPGAs, sugerindo uma arquitetura de software para integração das plataformas de software e o hardware usado para acelerar aplicações de simulação por dinâmica molecular. A proposta foi implementada como uma API para organização da comunicação entre as plataformas em níveis de abstração de serviço, visando tornar as camadas de software independentes do hardware.

Palavras Chave: API. Simulação por dinâmica molecular. FPGA. Aceleração por hardware.

A COMMUNICATION API FOR HARDWARE ACCELERATION OF MOLECULAR SIMULATIONS

ABSTRACT

The evolution of the integrated circuit manufacturing technology is still following the so called Moore Law. However, scientific applications growingly require high performance computational resources, motivating researchers to propose the acceleration of such applications through the use of dedicated hardware devices. Often, due to the need of obtaining fast results in the design of these applications the use of reconfigurable hardware devices is recommended.

Currently, there is a significant increase in the amount of research on molecular biophysics with a main goal on the design of drugs. Nonetheless, to achieve the design of a new drug and the possible cure of some disease, several complex procedures must be undertaken. As examples, it is possible to cite experiments to determine the behavior of simple molecules or proteins. Molecular dynamics simulations can reveal a large variety of facts about the molecular system under scrutiny. But to execute such simulations in a timely way, it is necessary to employ a huge amount of high performance computational resources, like supercomputers, large computer clusters or grids. This is due to the enormous amount of mathematical computations to perform, to the amount of generated information and to the need to obtain all this information in short time delays. This makes the requirement for high performance computing a basic characteristic of this field.

To fulfill the computational requirements of molecular dynamics simulations there are FPGA based platforms, which are frequently employed as hardware accelerators for applications with high computational cost. FPGAs are widely available and enable the fast design and implementation of dedicated hardware with high performance when compared to software running on general purpose processors.

The main contribution of this work is the proposition of a communication method between a host computer and a reconfigurable hardware platform based on FPGAs. The dissertation suggests a software architecture for integrating software and hardware platforms used to accelerate molecular dynamics simulation applications. The proposition has been implemented as an Application Programming Interface (API) that organizes the communication between platforms in several service abstraction levels, with the goal of rendering the application software layers independents of the accelerator hardware.

Keywords: API, Molecular dynamics simulation. FPGA. Hardware acceleration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ALPB	*Analytical Linearized Poisson-Boltzmann*
AMBER	*Assisted Model Building with Energy Refinement*
API	*Application Programming Interface*
CHARMM	*Chemistry at HARvard Macromolecular Mechanics*
CHARMMing	*CHARMM Interface and Graphics*
CI	*Controlador de Interface*
CPLD	*Complex Programmable Logic Devices*
DCM	*Digital Clock Manager*
DGEMM	*Dense Matrix Multiply Operation*
DM	*Dinâmica Molecular*
FF	*Flip-Flops*
FFT	*Fast Fourier Transform*
FIFO	*First In First Out*
FPGA	*Field-Programmable Gate Array*
GB	*Generalized Born*
GPL	*General Public License*
GPROF	*GNU Profiler*
GROMACS	*GROningen MAchine for Chemical Simulations*
GROMOS	*GROningen MOlecular Simulation*
HCPLD	*Highly Complex Programmable Logic Devices*
HDL	*Hardware Description Language*
HPC	*High Performance Computing*
HPRC	*High Performance Reconfigurable Computing*
HR	*Reconfigurable Hardware*
IC	*Interface de Comunicação*
IP	*Intellectual Property*
LAMMPS	*Large-scale Atomic/Molecular Massively Parallel Simulator*
LUT	*Look-Up Tables*
Mbps	*Megabits por segundo*
MD	*Molecular Dynamics*
MKL	*Math Kernel Lybrary*

MPD	*Management Processor Daemon*
MPI	*Message Passing Interface*
MPSoC	*MultiProcessor System on a Chip*
NAMD	*NAnoscale Molecular Dynamics*
NMR	*Nuclear Magnetic Resonance*
NoC	*Network on Chip*
PCH	*Parte Convertida em Hardware*
PDB	*Protein Data Bank*
PLL	*Phase Locked Loops*
PME	*Particle-Mesh Ewald*
PMEMD	*Particle Mesh Ewald Molecular Dynamics*
SANDER	*Simulated Annealing with NMR-Derived Energy Restraints*
SB	*Switch Box*
SoC	*System on a Chip*
SPLD	*Simple Programmable Logic Devices*
SPMD	*Single Program, Multiple Data*
SPME	*Smoothed Particle Mesh Ewald*
TMD	*Toronto Molecular Dynamics*
VHDL	*VHSIC Hardware Description Language*
VMD	*Visual Molecular Dynamics*

SUMÁRIO

1 INTRODUÇÃO

A bioinformática e a biofísica molecular computacional são áreas que necessitam de alto poder de processamento e beneficiam-se da disponibilidade de processamento paralelo tradicional HPC (do inglês, *High Performance Computing*). Há um grande número de aplicações nessas áreas que têm esta necessidade computacional. Uma destas aplicações são simulações de macromoléculas utilizando dinâmica molecular, uma técnica para simular o movimento e interações entre átomos ou moléculas usando as equações clássicas de movimento de Newton [YAN07]. Para simular macromoléculas e todos seus átomos ligados ou não-ligados, os pesquisadores precisam recorrer a recursos computacionais de alto desempenho [SCH05]. Por exemplo, em predição de estruturas de proteínas, alguns poucos nanossegundos de simulação podem exigir dias de processamento em um computador pessoal ou sobre uma grade computacional [BYS01]. Dependendo da estrutura a ser predita, podem ser necessárias centenas de nanossegundos simulados para tal macromolécula, inviabilizando a sua predição por meio de tais recursos computacionais[BAT07]. Em [ADC06], pode-se observar que mesmo em supercomputadores tem-se apenas um estudo de caso com o método PME [DAR93], definido na Seção 5.2, que ficou acima de 500 psec/Dia. Com isso, mostra a grande necessidade de avanços em HPC ou novas técnicas para aumentar o desempenho para esse tipo de aplicação, que envolve simulação por dinâmica molecular.

A combinação de microprocessadores convencionais e FPGAs (do inglês, *Field-Programmable Gate Arrays*) com diferentes níveis de granularidade produzem uma abordagem tecnológica denominada de HPRC (do inglês, *High Performance Reconfigurable Computing*). A granularidade é relativa ao tamanho, escala e nível de detalhamento de componentes, os dois principais tipos são: grão grande e grão pequeno. Em componentes de grão pequeno, pode-se aproveitar melhor o *hardware* para um determinado problema, utilizando técnicas e altos níveis de paralelismo, independente do tamanho dos dados ou da complexidade. Conseguem-se excelentes resultados de desempenho com FPGAs, por exemplo, que são baseados em grão pequeno [BEA06] [STR08] e agregado de máquinas com processadores de propósito geral é considerado grão grande.

Plataformas FPGAs são amplamente utilizadas para aceleração por *hardware* [AGA07b] [GU08] [SCR08] [YAN07], principalmente pela relativa facilidade em migrar partes de *software* para hardware por meio de compiladores de linguagens de alto nível para linguagem de descrição de *hardware* (HDL), obtendo-se flexibilidade na aceleração. Porém, na implementação direta em VHDL pode-se obter um melhor desempenho que na utilização de compiladores [SAH08], aumentando também a complexidade e o tempo de projeto. Nesta conversão de *software* para

hardware pode-se adicionar um maior nível de paralelismo na parte da aplicação que requer maior custo computacional, aumentando o desempenho global da aplicação.

Projetistas de sistemas digitais utilizando dispositivos reconfiguráveis (FPGA) constantemente pesquisam como melhorar o desempenho. A definição de uma plataforma de implementação é ponto importante para ser determinado. Quanto à plataforma, trata-se de um tipo de arquitetura a ser escolhida pelo projetista, dependendo dos recursos que serão utilizados na aceleração da aplicação. Assim, determina-se uma configuração adequada de hardware para a necessidade imposta na construção do projeto.

Aplicações que necessitam de HPC estão determinando uma nova finalidade ao desenvolvimento tecnológico que envolve as plataformas FPGAs (HPRC), mas ainda existem algumas limitações como demonstradas em [GU07]:

- Área do chip – Dependendo da aplicação podem-se citar alguns pontos que influenciam a área consumida no *chip*, como o tamanho do código, os cálculos, os tipos de variáveis (se utilizam ponto flutuante) e o grau de paralelismo ideal comparado com o que é viável.
- Projetistas – A complexidade das aplicações e a adequação ao hardware, o tempo de desenvolvimento de um projeto e a complexidade da programação em HDL.
- Lei de Amdahl – Diz que o desempenho máximo que pode ser encontrado em uma determinada melhoria está diretamente relacionado ao uso desta melhoria, assim não é possível diminuir o tempo não afetado pela melhoria. Um exemplo desta regra é a seguinte equação: 1/(F+(1-F)/P). Onde F é a fração de código sequencial, e (1-F) é a fração que pode ser paralelizada e P o número de processadores.
- Componentes – Restrição e acessibilidade em atributos chaves como multiplicadores e memórias.

Em cada equipamento HPRC realiza-se uma comunicação direta entre a máquina hospedeira e uma plataforma baseada em FPGA, pois a aplicação a ser implementada é dividida entre esses dois componentes determinados previamente. Com isto, há diversas APIs (do inglês, *Application Programming Interface*) abertas e privadas geralmente específicas para determinadas plataformas de *hardware* e aplicações científicas (*software*), limitando as possibilidades de portabilidade.

1.1 Modelo Conceitual da API

O principal objetivo desse trabalho é propor e implementar uma arquitetura de *software* para viabilizar a comunicação em aplicações que usam aceleração por hardware dedicado de simulações por dinâmica molecular. A comunicação nos casos específicos tratados aqui é realizada entre dois

componentes, o computador hospedeiro e uma plataforma baseada em FPGA. Ela deve se acontecer de forma transparente à aplicação, localizada no hospedeiro. O código da aplicação é modificado para conter chamadas à API, chamadas estas, sobretudo voltadas para a transferência de dados e comando de execução de tarefas pelo hardware dedicado. A arquitetura de *hardware* está sendo definida paralelamente a esse trabalho [MOH09], visando substituir uma parte da aplicação da máquina hospedeira por *hardware* dedicado com alto nível de paralelismo. O objetivo final é obter ganhos no desempenho global da aplicação.

O modelo conceitual da Figura 1 expõe a estrutura da API proposta e ilustra os componentes a serem utilizados para aceleração por *hardware* dedicado. Na máquina hospedeira será executada a aplicação que se deseja acelerar. Uma parte desta denomina-se PCH (Parte Convertida em *Hardware*), que possui maior custo computacional, será codificada em uma linguagem de descrição de *hardware*, por exemplo VHDL, e convertida em *hardware* no FPGA.

A API alvo do presente trabalho é uma arquitetura de *software*, transparente ao usuário da aplicação. Esta coleta informações relevantes para a PCH e as transfere pela IC (Interface de Comunicação) para a plataforma FPGA onde existe um Controlador de Interface (CI), que pode comportar ou não um processador, dependendo da interface a ser utilizada. Neste controlador reside uma parte da API, que em conjunto com a outra parte acoplada na aplicação terá a função de transferência de dados entre a máquina hospedeira e a plataforma FPGA. O PCH é o trecho da aplicação onde reside o maior custo computacional da aplicação e terá que ser previamente prototipado no FPGA antes da execução da aplicação. A conversão do PCH em *hardware* paralelizado no FPGA é objeto de um trabalho paralelo a esse [MOH09].

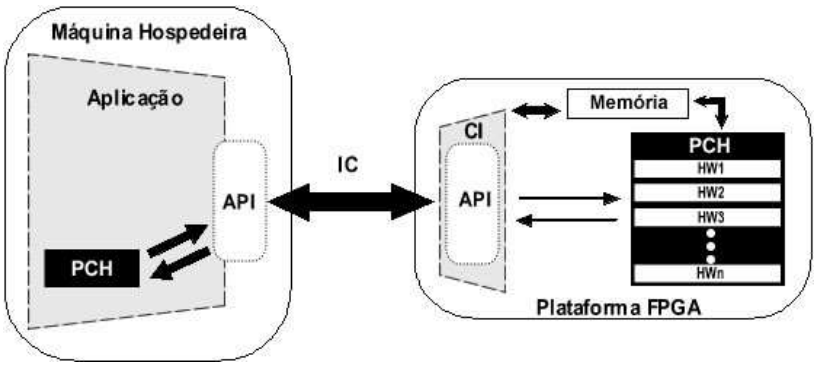


Figura 1 – Modelo conceitual da API proposta e os componentes envolvidos na máquina hospedeira e na plataforma FPGA.

O cenário apresentado nesta Seção é adequado para efetuar a aceleração por *hardware* de forma dedicada à aplicação. Com isso há a necessidade, em um primeiro momento, de efetuar a comunicação entre a máquina hospedeira e a plataforma FPGA para contribuir na aceleração total da aplicação.

1.2 Importância do uso de APIs em Aceleração por Hardware

Existem diversos fabricantes de arquiteturas HPRC. Durante o uso de *hardware* reconfigurável, torna-se necessário o uso de *software* do próprio fabricante para comunicação entre a máquina hospedeira e a plataforma FPGA e a existência de um ambiente de desenvolvimento de *hardware*, tipicamente baseado em uma linguagem HDL. Porém, com o grande número de fabricantes de plataformas FPGAs de baixo custo, acoplado a aplicações executadas em uma máquina hospedeira, surge a necessidade de APIs para conectar essas máquinas hospedeiras e as plataformas baseadas em FPGAs. Isto viabiliza a utilização em diferentes plataformas de HPRC, trazendo portabilidade e diversidade as plataformas FPGA empregadas.

Devido ao tempo significativo de desenvolvimento de um projeto de HPRC, deve-se fazer uso intensivo da técnica de reutilização de componentes. Assim, a confecção de uma API portável e de fácil reuso está diretamente relacionada ao tempo total do projeto de uma plataforma HPRC. Para isso, conta-se com a utilização de linguagens de programação padronizadas e portáveis. O emprego de uma API em diferentes projetos de HPRC é assim facilitado, desde que esta seja portável e reutilizável.

1.3 Objetivos do Trabalho

A proposta desse trabalho é de construir uma arquitetura de *software* para a comunicação da máquina hospedeira com a plataforma FPGA. Na máquina hospedeira, o *software* será acoplado à aplicação de simulação por dinâmica molecular, visando à transferência de dados desta aplicação para a plataforma de *hardware* utilizando uma interface de comunicação.

1.4 Organização do Restante do Documento

O restante desse trabalho está organizado em dez Capítulos, como descrito a seguir.

O Capítulo 2 apresenta um referencial teórico que servirá de base para o entendimento de diversas partes posteriores no trabalho e tornando mais fácil a leitura.

O Capítulo 3 descreve uma revisão do estado da arte em APIs aplicadas ao uso de FPGAs e aceleração por *hardware* dedicado.

O Capítulo 4 traz informações sobre materiais e métodos utilizados nos experimentos que serão descritos nos Capítulos seguintes.

No Capítulo 5 aborda os estudos de casos realizados e a comunicação por meio de troca de mensagens. A Seção 5.1 apresenta-se uma análise do código do *software* PMEMD, visando a proposta da API. Uma ferramenta de traçado de perfil de execução (**`gprof`** da GNU) é empregada

para definir e quantificar o comportamento do *software* PMEMD e os tempos consumidos na execução das rotinas que compõe esta aplicação. Na Seção 5.2 complementa a compreensão das aplicações de simulação por dinâmica molecular, pela discussão de experimentos realizados com duas aplicações e quantidades variadas de processos. Os conceitos básicos relacionados à comunicação quando se usa processamento paralelo baseado na biblioteca clássica MPI está exposto na Seção 5.3. Adicionalmente, apresenta-se outra contribuição desse trabalho, ao decidir pelo emprego ou não da biblioteca MPI em conjunção com plataformas FPGA.

As dificuldades encontradas na implementação e os recursos utilizados na elaboração da arquitetura de *software* são mostrados no Capítulo 6, incluindo os recursos de *hardware* e *software* escolhidos conforme a necessidade do trabalho.

Uma proposta da arquitetura de *software* ou API é apresentada no Capítulo 7. Define-se aqui as rotinas a serem padronizadas e implementadas na API, também é demonstrado dois experimentos realizados com a proposta, trazendo resultados preliminares da arquitetura de *software*.

O Capítulo 8 conclui esta Dissertação e identifica aspectos a serem abordados em trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste Capítulo apresentam-se alguns conceitos importantes para o entendimento desse trabalho e componentes utilizados ou agregados à implementação da API no contexto de simulações de dinâmica molecular com *hardware* reconfigurável.

2.1 Dispositivos Lógicos Programáveis

Circuitos integrados digitais implementados em uma pastilha de silício podem ser classificados conforme a disponibilidade desses CIs, como padrão ou de prateleira (do inglês, *off-the-shelf*), e para uma dada aplicação específica (em inglês, *Aplication Specific Integrated Circuits* ou ASICs). Circuitos integrados digitais são constituídos por portas lógicas e necessitam de outros componentes para realizar uma função específica, para determinar as funcionalidades ao projeto de *hardware*.

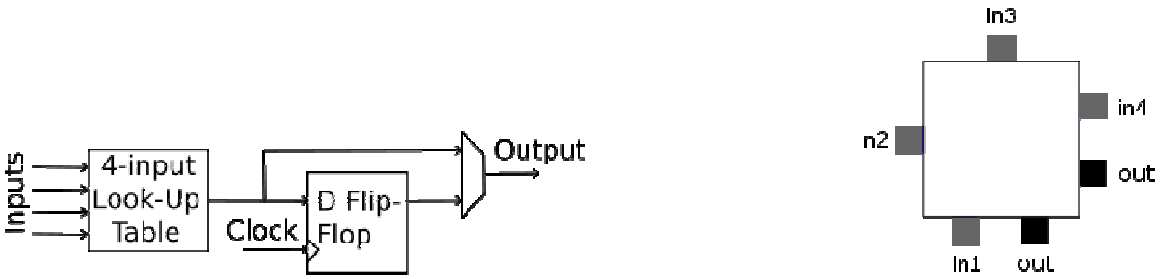
Dispositivos lógicos programáveis ou PLDs (do inglês, *Programmable Logic Devices*) são circuitos integrados que não possuem uma funcionalidade fixa, podendo ser reconfigurados pelo usuário quando houver necessidade, facilitando as alterações no projeto. Uma característica destes é a capacidade de programação das funções lógicas pelo usuário. Conforme [COS06], PLDs podem ser classificados em função da capacidade de portas lógicas equivalentes, e os principais grupos nesta categoria são:

- SPLDs (*Simple Programmable Logic Devices*): são dispositivos simples de baixa capacidade, para pequenos projetos; geralmente contêm menos de 600 portas lógicas.
- HCPLDs (*Highly Complex Programmable Logic Devices*): são dispositivos de alta capacidade e complexidade, para grandes projetos, geralmente contêm mais de 600 portas, atualmente podendo chegar às dezenas de milhões de portas, e englobam os dispositivos CPLDs (*Complex Programmable Logic Devices*) e FPGAs (*Field Programmable Gate Arrays*).

2.1.1 FPGAs

FPGAs são circuitos lógicos que consistem em um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. Cada bloco lógico contém tipicamente capacidade computacional para implementar um conjunto de funções lógicas, realizar roteamento para comunicação entre blocos e alguma capacidade de armazenamento de informação. Os blocos lógicos consistem de LUTs (do inglês, *Look-Up Tables*) e FFs (do inglês, *Flip-Flops*)

conforme Figura 2(a). Já na Figura 2(b), mostra-se uma interface típica de entradas e saídas de um bloco lógico, mostrando a localização dos pinos de entrada e saída. As LUTs possuem várias entradas (existem dispositivos que usam LUTs de 4 entradas, outros de 6 entradas e alguns usam LUTS de vários tipos) e dependendo do fabricante ou modelo do FPGA as LUTs podem ocorrer em quantidades e ter características diferentes.



(a) (b)

Figura 2 – Bloco Lógico: (a) Exemplo de um bloco configurável, contendo uma LUT de 4 entradas um Flip-Flop e circuito de escolha para gerar a saída (Multiplexador 2:1); (b) Localização típica de pinos de entrada e saída para um bloco configurável, com quatro entradas e duas saídas.

Algumas características de arquiteturas de FPGAs podem variar dependendo do fabricante, conforme exemplifica a

Tabela 1, onde se mostra as diferenças entre três dos principais fabricantes de FPGAs a partir de uma das famílias de dispositivos de baixo custo que estes oferecem. A referência [DUB08] define as características típicas de FPGAs como:

- Módulos de Propriedade Intelectual (do inglês, *Intellectual Property Cores* ou *IP Cores*): são módulos de *hardware* acoplados à matriz básica de elemento reconfiguráveis como multiplicadores dedicados e memória embutida, suprindo recursos não passíveis de implementação como os elementos da matriz ou onde uma implementação seria muito custosa usando estes elementos. Tais módulos costumam ser eles mesmos reconfiguráveis em algum grau. Um exemplo é a possibilidade de escolher a largura de acesso à memória em bit, byte ou palavras de 16 ou 32 bits.
- Distribuição de recurso de relógio: Gerenciamento digital de relógio (do inglês, *Digital Clock Manager* ou DCM) ou analógicos (denominados em inglês *Phase Locked Loops* ou PLLs) suprindo controle de frequência e escorregamento do(s) sinal(is) de relógio.
- Características de E/S: permite a escolha de um dentre uma gama de padrões de E/S comerciais tais como TTL, CMOS, ou padrões diferenciais (que usam dois pinos por bit de informação transmitida).
- Permitir que processadores com diversos graus de complexidade sejam

implementados ou disponibilizados em um FPGA. Quando se implementa processadores usando os elementos da matriz básica de elementos reconfiguráveis diz-se serem estes *soft IP cores* ou *soft processors*. A outra possibilidade é o fabricante disponibilizar processadores sob a forma de um IP Core dentro do FPGA, os chamados *hard IP cores* ou *hard processors*. Exemplo de dispositivo com tais módulos são vários membros da família VirtexII-Pro da Xilinx, que possuem de um a quatro processadores PowerPC 405, um processador RISC de 32 bits.

Tabela 1 – Exemplos de diferenças nas características arquiteturais de FPGAs, dependendo do fabricante. Apenas uma família de cada fabricante é apresentada [DUB08].

Características	Xilinx SPARTAN 3TM	Altera Cyclone III	Actel Fusion®
Blocos lógicos (sequenciais e combinacionais)	Células Lógicas (Logic cell (LC))	Elementos Lógicos (Logic element (LE))	Elementos Lógicos (Logic element)
Memória Embarcada	Block RAM	RAM blocks	RAM blocks
Fios Globais de Relógio	Sim	Sim	Sim
Multiplicadores em Hardware	Sim	Sim	Sim
Tecnologias de Programação	SRAM	SRAM	Flash
ADC Integrado e driver MOSFET	NA	NA	Sim

Em [ORD06] apresenta-se as quatro principais organizações internas utilizadas em circuitos programáveis: Matriz simétrica, *Sea-of-gates*, *Row-based* e PLD hierárquico. A matriz simétrica é bastante difundida, pela flexibilidade no roteamento e por possuir canais horizontais e verticais. Sua estrutura básica é formada por blocos lógicos, blocos de entrada e saída e segmentos de fios, como pode ser observado na Figura 3.

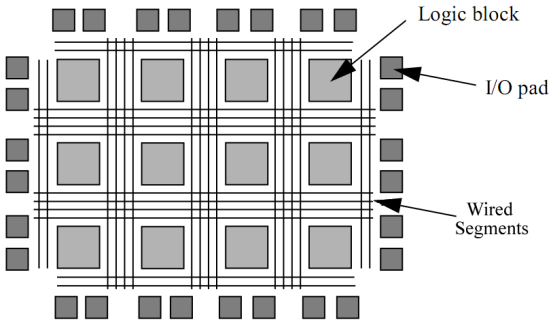


Figura 3 – Arquitetura FPGA do tipo matriz simétrica. Contêm blocos lógicos, blocos de entrada e saída e segmentos de fios [AHM00].

Para interconectar os segmentos de fios ou canais horizontais e verticais existem blocos chamados de caixas de comutação ou SBs (do inglês, *Switch Boxes*). Essas estruturas podem ser visualizadas na Figura 4, organizadas em uma arquitetura de matriz simétrica e um detalhamento de sua funcionalidade interna para interconectar segmentos de fios que chegam a sua interface.

Dentre as diversas evoluções de FPGAs complexos, uma que se destaca é a agregação de blocos lógicos de base em *clusters* lógicos, que criam uma região de alta conectividade com mais de

um bloco lógico. Em [AHM00] e [BET98], mostra-se que a utilização de clusters é vantajosa para a eficiência em área e o desempenho do FPGA, por diminuir atrasos, sobretudo de fios longos. Diversas outras evoluções de FPGAs podem ser encontradas na literatura atual.

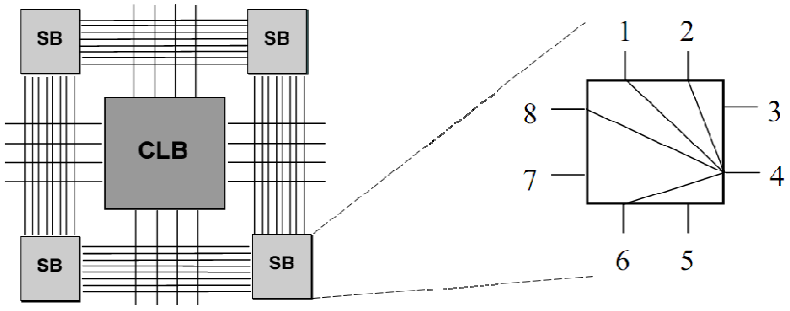


Figura 4 – Caixa de Comutação (Switch Box). À direita nota-se que o fio que entra no pino 4 da SB pode ser conectado a um ou mais fios em cada um dos outros lados da SB. NO caso, mostra-se 4 possibilidades.

FPGA é um tópico de pesquisa diretamente vinculado a produtos industriais que trazem benefícios econômicos a muitas empresas. Assim, existe uma imensa quantidade de trabalhos abordando o projeto e o emprego desses dispositivos, tanto na área acadêmica como na área industrial.

2.2 Dinâmica Molecular

A mecânica molecular usa funções simples de energia potencial (tais como oscilador harmônico ou os potenciais de Coulomb) para modelar sistemas moleculares. Esta mecânica é largamente aplicada no refinamento de estruturas moleculares e em simulações por dinâmica molecular (do inglês *molecular dynamics* ou MD), Monte Carlo (MC), ou *ligand-docking* [ADC06]. Esse refinamento é o passo inicial na fase de preparação do sistema molecular que posteriormente será submetido a simulações por dinâmica molecular.

O método por dinâmica molecular é amplamente utilizado para obter informações como a evolução conformacional, de cinética e termodinâmica em proteínas e outros sistemas moleculares ao longo do período de simulação. Ainda, podem-se obter detalhes mais precisos a respeito do movimento de partículas individuais em função do tempo.

Em simulações por dinâmica molecular existem três características básicas necessárias à execução desse método: (i) um modelo para interação entre os elementos do sistema (átomos, moléculas, vértices, etc.); (ii) um integrador para propagar as posições das partículas e velocidades em um determinado tempo; (iii) e a escolha de um conjunto (do inglês, *ensemble)* estático para definir parâmetros como pressão, temperatura, volume, entre outros [SUT02].

2.2.1 Interação entre os Elementos do Sistema

A dinâmica molecular e outras técnicas são relacionadas em uma metodologia de simulação computacional integrando a equação de movimento. Desta forma, o modelo da evolução no tempo em um grupo de interações entre átomos ou moléculas determina a força em cada átomo, repetindo essas interações entre todos os átomos, gerando a evolução do sistema biológico. A dinâmica molecular é baseada em mecânica clássica, ou seja, na segunda lei de Newton (F=ma) [AGA07b].

A força em cada átomo é obtida por intermédio da derivada do potencial relacionado com suas coordenadas. Com essa força e as equações de movimento pode-se descrever como as coordenadas atômicas variam com o tempo, e em cada passo da dinâmica essas forças são reavaliadas.

A função da energia potencial é dividida em duas partes: átomos ligados e não ligados, conforme a Equação 1. Na primeira parte, átomos ligados, contêm os cálculos do número de ligações covalentes, dos ângulos e dos diedros. Já a parte de átomos não ligados possui os cálculos de Van Der Waals e as interações eletrostáticas [AGA07b].

$$E(potencial) = \sum_{ligações} E_l + \sum_{ângulos} E_a + \sum_{dihedros} E_d + \sum_{j=1}^{N}\sum_{i=1}^{N}(E_{vdw}) + \sum_{j=1}^{N}\sum_{i=1}^{N}(E_{el}) \quad ...(1)$$

Equação 1 – Cálculo da Energia Potencial – a primeira parte, com três termos, está relacionada aos cálculos de átomos ligados, e a segunda, com dois termos, a átomos não ligados [AGA07b].

A Figura 5 apresenta os tipos de forças que ocorrem entre átomos ligados.

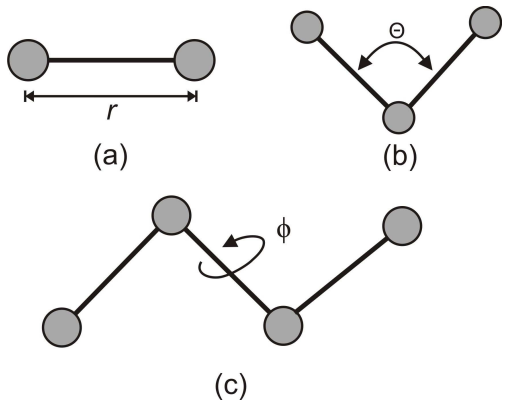


Figura 5 – Representação das grandezas entre átomos ligados: (a) Ligação covalente entre pares de átomos; (b) Ângulo entre os átomos e suas respectivas ligações covalentes; (c) Ângulo de torção entre os átomos e suas ligações covalentes.

Nos átomos ligados, o primeiro termo contém os cálculos do potencial em ligações covalentes entre pares de átomos baseado principalmente na distância e no tipo de ligação química, conforme Figura 5(a). No segundo termo, associam-se ao cálculo as alterações angulares entre pares

de ligações como mostra a Figura 5(b). Assim como as ligações químicas, as oscilações dos ângulos podem ser descritas por um potencial harmônico. No último termo de átomos ligados, representa-se o ângulo ϕ de torção entre átomos separados por três ligações covalentes com os cálculos dos diedros entre estes, como ilustrado na Figura 5(c).

A Equação 1 pode ser reescrita com os parâmetros matemáticos corretos de cada termo original mostrado na Equação 2. Além disso, apresenta-se todos os termos separadamente do cálculo de energia potencial com suas respectivas definições, seja de átomos ligados ou não ligados definidos em [AMB08a].

$$
\begin{aligned}
U(R) = & \sum_{ligações} K_r\left(r - r_{eq}\right)^2 && \text{ligações} \\
& + \sum_{ângulos} K_\theta\left(\theta - \theta_{eq}\right)^2 && \text{ângulos} \\
& + \sum_{dihedros} \frac{V_n}{2}\left(1 + \cos\left[n\phi - \gamma\right]\right) && \text{diedros} \\
& + \sum_{i<j}^{átomos} \frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} && \text{van der Waals} \\
& + \sum_{i<j}^{átomos} \frac{q_i q_j}{\varepsilon R_{ij}} && \text{eletrostática} \qquad ...(2)
\end{aligned}
$$

Equação 2- Cálculo da Energia Potencial com toda descrição matemática de seus termos.

Cálculos envolvendo termos de átomos ligados são extremamente rápidos em simulações computacionais. Porém, os cálculos de átomos não ligados possuem complexidade assintótica $O(N^2)$, onde N é o número de átomos da molécula, que pode chegar a centenas de milhares em um sistema normal e a milhões em grandes sistemas biológicos.

O cálculo de Van Der Waals é a implementação do potencial de Lennard-Jones, quarto termo da Equação 2. Este utiliza funções de atração e repulsão dos átomos, dependendo da distância e de constantes determinadas experimentalmente, expressando a natureza e a interação entre os átomos. O raio de corte ou *cutoff* pode ser configurado na descrição da dinâmica molecular, variando-o conforme a necessidade do usuário, assim determina-se a área de atuação do pequeno alcance ou *short-range* servindo como um parâmetro de limitação do seu alcance. Já nos cálculos de longo alcance ou *long range*, utiliza-se o potencial de Coulomb, onde todos os átomos interagem entre si pelas nuvens eletrônicas, quinto termo da Equação 2 [SUT02]. Quando os átomos estão muito próximos entre si, a força repulsiva cresce rapidamente e o ponto de equilíbrio é dado pelo raio de Van Der Waals.

O somatório de Ewald é limitado por condições periódicas, esse método considera o potencial devido às modificações parciais do sistema, junto com todas as suas imagens periódicas

pelo uso da decomposição. Divide-se uma interação de Coulomb em um termo de pequeno alcance, manipulando exatamente uma soma direta, mais um longo alcance, variando suavemente o termo e tratando a aproximação na soma recíproca por meio de métodos de Fourier [DAR99]. Desta forma, torna-se o cálculo de pequeno alcance o recurso mais utilizado, determinando a maior parte do tempo da execução da simulação por DM, e com maior custo computacional. Na Seção 5.1 mostra com maiores detalhes a função *short_ene,* responsável pelos cálculos de pequeno alcance.

O método somatório de Ewald é amplamente utilizado na manipulação das interações de Coulomb para sistemas periódicos, esse método é representado pelas interações eletrostáticas contidas na Equação 2. Estas convertem séries convergentes em uma soma de constantes do espaço real e o recíproco. O método PME [DAR93] reduz a complexidade da soma de Ewald, onde o espaço real é computado em $O(N)$ e o recíproco passa de $O(N^2)$ para $O(N\log N)$ usando respectivamente as técnicas denominadas *particle-mesh interpolation procedure* e a já conhecida FFT (do inglês, Fast Fourier Transform). Em trabalhos recentes, estas técnicas são amplamente aplicadas por essa otimização na complexidade assintótica desses cálculos [ADC06] [AGA07a] [AGA07b] [DAR93] [LIU04].

2.2.2 Integrador

Para sistemas com mais de dois átomos a resolução da equação de movimento de Newton precisa de uma solução por meio de métodos numéricos [STO09]. O integrador traz características físicas do sistema ao modelo potencial, dando uma maior precisão nos resultados da simulação. Quando não há erros de simulação o integrador proporciona resultados exatos dentro do modelo, caso contrário é feita uma aproximação para o sistema com desenvolvimento contínuo no tempo, por se tratar de sistemas periódicos [SUT02]. Conforme [SUT02], o integrador deve obedecer alguns requisitos como:

- Precisão: No sentido em que se aproxima muito de uma trajetória verdadeira.
- Estabilidade: No sentido em que conserva energia e que as pequenas perturbações não levam a instabilidades no sistema.
- Robustez: No sentido em que se permitem grandes passos de tempo ou *time steps*, a fim de uma propagação eficiente do sistema pelo espaço.

Existem diversos tipos de integradores os três principais são os integradores baseados na expansão de Taylor, que são integradores de um operador baseado em métodos de divisão que possibilitam integração de movimento e integradores que consideram o grau de liberdade das moléculas [SUT02].

Como as simulações dos sistemas moleculares tentam resolver equações matemáticas há a

necessidade de utilização de métodos numéricos para inicialmente aproximar essas equações usando, por exemplo, a série de Taylor. Os métodos numéricos, possuem erros de truncamento e erros de aproximação. A aproximação das equações matemáticas introduz o chamado erro de truncamento e os erros de aproximação ocorrem pelo fato de que os computadores representam os seus dados numéricos com limitações de dígitos. Então, os integradores para obedecer aos seus requisitos de precisão e devem contabilizar, além das aproximações, os erros gerados por esta integração baseados em estimativas de erros.

A energia potencial é uma função contínua das posições e onde o passo de tempo ou *time step* seja pequeno o suficiente para se considerar que as posições variem suavemente com o tempo, em um dado conjunto de posições atômicas em um determinado instante, as posições no próximo passo podem ser obtidas por uma expansão de Taylor [MUN09]. A forma mais simples e direta de construir um integrador é pela expansão das posições e velocidades em uma série de Taylor. Um popular algoritmo que utiliza séries de Taylor é o de Verlet, que faz a integração das equações de movimento de Newton, além disso, é usado para calcular as trajetórias para cada átomo e em cada incremento de tempo em simulações por dinâmica molecular. Conforme [STO09], além do algoritmo de Verlet, existem diversos outros algoritmos numéricos para integração de equações de movimento: Algoritmo Leap-Frog, Velocity Verlet, Algoritmo de Beeman.

2.2.3 Conjunto Estático

As condições experimentais são importantes para trazer precisão na simulação, pois existem valores para condições físicas, pressão e temperatura, que são replicados e devem ser observados na simulação. Um conjunto ou *ensemble* é uma coleção de um grande número de sistemas possíveis que possuem diferenças ou parâmetros a serem definidos de forma microscópica e que pertencem a sistemas macroscópicos ou termodinâmicos. Cada um dos sistemas em um determinado conjunto com N interações entre átomos ou moléculas, possuem valores pré-determinados conforme o tipo de conjunto utilizado. Existem diferentes tipos de conjuntos com suas próprias características, porém os mais usados em simulações por dinâmica molecular são os seguintes [ADC06]:

- O conjunto canônico (NVT): Com o número de átomos fixos (N), volume fixo (V) e temperatura fixa (T).
- O conjunto isobárico-isoentalpia (NPH): Com o número de átomos fixos (N), pressão fixa (P) e entalpia fixa (H).
- O conjunto isobárico-isotérmico (NPT): Possui o número de átomos fixos (N), pressão fixa (P) e temperatura fixa (T).
- O grande conjunto canônico (μVT): Com potencial químico fixo (μ), volume fixo e

temperatura fixa.

- O conjunto microcanônico (NVE): Possui o número de átomos fixos (N), volume fixo (V) e energia fixa (T).

Outra propriedade importante no ambiente de simulação por dinâmica molecular é a solução química adequada ao sistema molecular. As soluções implícitas é um tipo de solvente utilizado em ambientes de simulação por DM para que se criem dinamicamente soluções, de água pura ou contendo íons, durante a simulação do sistema molecular. Porém, parte do tempo computacional da simulação é gasto com interações entre solventes, isto pode ser evitado com a utilização de soluções explícitas baseadas em água quando possível, que são adicionadas antes de iniciar a simulação na fase de preparação do sistema molecular. Existem vários tipos de modelos acessíveis de águas explícitas, para que o pesquisador possa agregar em seu sistema molecular esse tipo de solvente para trazer um comportamento adequado do sistema molecular imerso. Os tipos mais populares de águas explícitas são: TIP3P, TIP4P, TIP5P, SPC e SPC/E [ADC06].

O foco desse trabalho está direcionado aos cálculos de átomos não ligados que contêm os cálculos de van der Waals e de interações eletrostáticas, onde se concentra o maior custo computacional em toda a simulação, determinando a sua complexidade e seu potencial para recursos de HPRC.

2.3 Aplicações de Simulação por Dinâmica Molecular

Diversos grupos de pesquisa de biofísica molecular dispõem de recursos computacionais de alto desempenho para investigação de sistemas biológicos utilizando aplicações de simulação por dinâmica molecular. Existem disponíveis aplicações baseadas nos termos de *software* livre e proprietárias com esta função, os mais populares sendo: AMBER [CAS05][PEA95], CHARMM [BRO83], DL_POLY [SMI96][TOD04], GROMACS [BER95][HES08][LIN01], LAMMPS [LAM08][PLI95], NAMD [KAL99][PHI05] e Protomol [MAT01].

Programas de simulação por dinâmica molecular simulam o comportamento de sistemas biomoleculares pela sua evolução temporal. Esses comportamentos são do movimento contínuo dos átomos, da vibração das ligações químicas, da variação dos ângulos dessas ligações e da rotação da molécula.

Um grande desafio científico está na exploração efetiva de todos os recursos (Processador, Memória e I/O) de plataformas HPC. A utilização de máquinas paralelas é popular nesse tipo de aplicação. De acordo com [HEI05], existem dois principais métodos para se paralelizar códigos em aplicações por dinâmica molecular: o método de replicação de dados e o método de decomposição de domínios.

Método de replicação de dados

É o método em que todos os nodos de processamento mantêm os dados de coordenadas e forças de todo o sistema, e a paralelização é obtida pelo algoritmo de decomposição de forças ou de partículas. Quando surge a necessidade de atualização dos dados, tem que ser realizado em todos os nodos. Com isso, dependendo da aplicação, pode-se gerar um grande volume de dados, pois em cada nodo de processamento terá que ter uma réplica dos dados. Assim, o número de operações de envio e recepção (E/R) desses dados pode-se tornar um problema se a quantidade de atualizações for alta.

Método de decomposição de domínios

O princípio desse método é designar geometricamente domínios para diferentes processadores. Assim, as partículas não têm ligações longas para certo processador, mas pode ser transferido de um processador a outro, ajustando para as posições espaciais deles. Esse método é designado para sistemas com interações de pequeno alcance ou onde tem que ser aplicado o raio de corte no espaço [SUT02]. Na utilização desse método diminui-se a quantidade de operações de envio e recepção (E/R), por cada nodo manter apenas uma parte das posições e coordenadas atômicas. A Figura 6 demonstra a comunicação desse método em 2 dimensões, que escolhe quatro processadores de borda colocando-se rótulos e iniciando a sequência para comunicação. Primeiro as informações são E/R da esquerda e direita armazenando as coordenadas de três processadores em cada um. Depois as informações são E/R para cima e para baixo finalizando o processo de comunicação [SUT02].

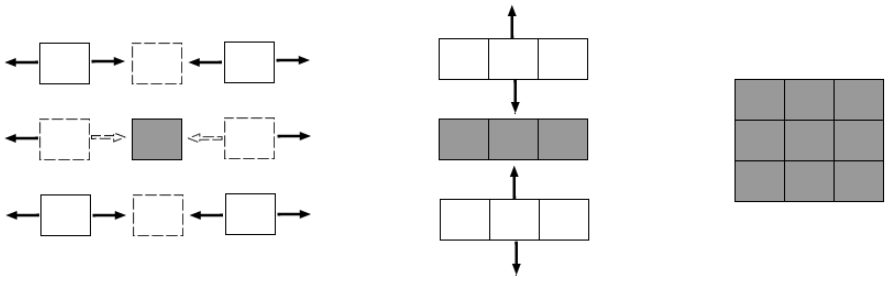


Figura 6 – Forma de comunicação entre os processadores no método de decomposição de domínio em 2 dimensões [SUT02].

O método de decomposição de domínio possui maior escalabilidade para grande quantidade de processadores e sistemas biológicos mais complexos, como se pode constatar na leitura de [HEI05] [PLI95]. Porém, em [AGA07b], discutido no Capítulo 3, mostra-se que é possível acelerar aplicações de simulação por DM que aplicam o método de replicação de dados por meio de *hardware* reconfigurável, conseguindo obter sucesso na aceleração, como mostrado para o pacote AMBER.

2.3.1 AMBER

A suíte AMBER [CAS05] [PEA95] (do inglês, *Assisted Model Building with Energy Refinement*) é uma coleção de programas que realiza os três passos principais em processamento de dados de sistemas biológicos: Preparação, Simulação e Análises de Trajetórias. Pode-se observar o fluxo de dados nesta suíte na Figura 7.

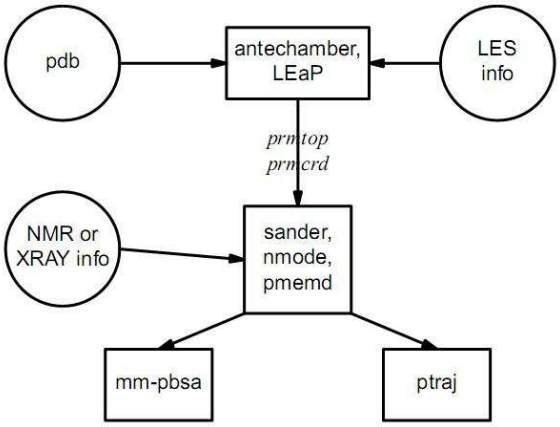


Figura 7 – Programas utilizados em cada passo do pacote AMBER. Na preparação por meio do arquivo pdb(Protein Data Bank), tem-se o *antechamber* e o LEaP gerando os arquivos prmtop e prmcrd, que possuem todas as características da molécula como a disposição dos átomos e suas ligações, as suas coordenadas no espaço, os seus ângulos, entre outros. Estes arquivos são incluídos na entrada dos simuladores onde se pode contar com três programas o SANDER, NMODE e PMEMD; por último, no passo de análise de trajetórias dos átomos encontram-se o mm-pbsa e o ptraj que são responsáveis por esta função [AMB08a].

O código da suíte AMBER é baseado em linguagem C e Fortran [AMB08a]. O Fortran ainda é uma linguagem muito popular em pesquisas científicas principalmente para cálculos na área de física. Nas primeiras versões do AMBER 30% do código era escrito em linguagem C [PEA95].

A execução do AMBER em modo paralelo é determinada pelo método de replicação de dados por intermédio de uma interface de troca de mensagens entre os processadores participantes do anel que compõem o cluster.

SANDER (do inglês, *Simulated Annealing with NMR-Derived Energy Restraints*) é um programa de simulações por dinâmica molecular, que gera a saída da minimização de energias e refinações NMR (do inglês, *Nuclear Magnetic Resonance*) baseadas em restrições de distâncias, torções de ângulos e penalidades de funções em mudanças químicas [AMB08a]. Esse *software* trabalha com toda a estrutura da molécula pelas interações entre os seus átomos. Ele utiliza equações newtonianas de movimento, relacionadas com fatores de pressão, temperatura e suas coordenadas no espaço. O objetivo é calcular a energia gerada em cada átomo e conseqüentemente a energia potencial total do sistema biológico.

Encontra-se na suíte AMBER uma otimização do *software* SANDER, chamado de PMEMD

(do inglês, *Particle Mesh Ewald Molecular Dynamics*). Este reimplementa o SANDER com o objetivo de aumentar o desempenho em simulações por dinâmica molecular. O PMEMD utiliza processamento paralelo e foi reescrito todo em Fortran 90 com capacidade de executar sobre os mesmos arquivos de entrada utilizados no SANDER em versões posteriores à 6 [AMB08b].

PMEMD é um subconjunto do SANDER. No início, esse sistema era utilizado apenas para dar suporte a simulações com o PME (do inglês, *Particle Mesh Ewald*), por isso o nome PMEMD. Em versões posteriores [AMB08a] foram adicionadas mais funcionalidades como simulações com GB (do inglês, *Generalized Born*) e ALPB (do inglês, *Analytical Linearized Poisson-Boltzmann*).

2.3.2 CHARMM

O CHARMM (do inglês, *Chemistry at HARvard Macromolecular Mechanics*) é uma ferramenta de pesquisa para biologia computacional desenvolvida na Universidade de *Harvard*. Esse *software* é um pacote para simulações utilizado principalmente no estudo de proteínas, estrutura de ácidos nucléicos e funções, especialmente com macromoléculas de proteínas, DNA, RNA e outros sistemas biológicos complexos. Além disso, é utilizado para diversas funções, como: simular dobramentos reversíveis de estruturas peptídicas, determinar o dobramento de energias livres, em métodos de mecânica clássica para investigar energias potenciais, investigar processos químicos, entre outros [TAU02]. A aplicação mais comum do CHARMM é dinâmica molecular.

Atualmente, o CHARMM criou um portal como ferramenta para prover uma interface amigável para o pacote e facilitar a preparação, execução e visualização de simulações moleculares, chamado de CHARMMing (do inglês, *CHARMM interface and graphics*) [MIL08].

Conforme [NAT04], a execução paralela do CHARMM compartilha todas as forças e coordenadas em todos os processadores, e a maior eficiência em paralelo é adquirida com 16 processadores. Com isso, determina-se o método empregado que é o de replicação de dados pela similaridade nas execuções em paralelo com as descritas em outras aplicações, por exemplo, o AMBER.

2.3.3 DL_POLY

Recentemente, surgiu uma iniciativa do Laboratório Daresbury em desenvolver um pacote para simulações macromoleculares chamado de DL_POLY. Trata-se de um pacote de propósito geral para simulação por dinâmica molecular. O objetivo é aplicar métodos por dinâmica molecular em paralelo para sistemas complexos, em especial macromoléculas. Inicialmente foi utilizado o método de replicação de dados para efetuar a paralelização desta aplicação [SMI96].

Na versão mais recente (3), do DL_POLY, utiliza-se uma estratégia de paralelização por

decomposição de domínios com mais eficiência e escalabilidade. Houve uma nova adaptação da decomposição de domínios com o método SPME (do inglês, *Smoothed Particle Mesh Ewald*) [ESS95], considerado uma atualização do PME, para calcular forças de longa distância em simulações moleculares agregadas a FFTs (do inglês, *Fast Fourier Transform*) de três dimensões. Com estas técnicas é possível fazer simulações de sistemas na ordem de um milhão de partículas em diante, considerados como grandes sistemas biológicos [TOD04].

2.3.4 GROMACS

É um pacote para simulação por dinâmica molecular desenvolvido originalmente na University of Groningen e possui licença GPL (do inglês, *General Public License*). O GROMACS (do inglês, *GROningen MAchine for Chemical Simulations*) é uma coleção de programas para simulação por dinâmica molecular e análise de trajetória de dados. É popular em computadores paralelos, por possuir eficiência nas implementações paralelas de propósito gerais, com códigos utilizando dinâmica molecular. O GROMACS é baseado no *software* GROMOS (do inglês, *GROningen MOlecular Simulation*), referência para algoritmos de simulação, enquanto que a paralelização utiliza métodos propostos em literatura mais recente [LIN01].

Nas primeiras versões do GROMACS, escrito em linguagem ANSI C o paralelismo era baseado na decomposição de partículas. A comunicação entre os processadores foi limitada pela distribuição de forças e posições sobre o anel do cluster e realizado uma vez a cada passo de tempo (*time-steps*) da simulação [BER95].

Atualmente, no GROMACS 4, estão presentes diversas atualizações: em ferramentas acopladas, alto desempenho em processadores simples, na otimização de algoritmos, e em máquinas paralelas com algoritmo de decomposição de domínios, balanceamento de carga e redução na comunicação [HES08]. Com isso, pode ser adotado em sistemas moleculares complexos atuando de forma mais eficiente em máquinas paralelas.

2.3.5 LAMMPS

Um pacote clássico para DM é o LAMMPS (do inglês, *Large-scale Atomic/Molecular Massively Parallel Simulator*). Trata-se de um código desenvolvido para simulação molecular e sistemas atômicos em computadores paralelos usando técnicas de decomposição espacial, que decompõe o domínio da simulação em pequenas partes tri-dimensionais. Desenvolvido pelo *Sandia National Laboratory*, possui licença GPL e contém três versões principais. A primeira, de 1999 é escrita em código Fortran 77. Em 2001 o código foi melhorado para efetuar gerenciamento de memória em código Fortran 90. A última versão foi reescrita em C++ em 2004. Todas as versões

realizaram a comunicação através de MPI (do inglês, *Message Passing Interface*). Porém, diversas mudanças foram feitas nos algoritmos paralelos, especialmente em cálculos de átomos não ligados [LAM08] [PLI97].

2.3.6 NAMD

Simulação por DM contém uma enorme complexidade computacional e as máquinas paralelas provêem um potencial para este desafio computacional. Existe a necessidade, além de recursos computacionais, de desenvolver programas escaláveis e de fácil modificação pelos programadores. NAMD2 (do inglês, *NAnoscale Molecular Dynamics*) é um programa que provê estas características. Esse aplicativo é um código paralelo que utiliza dinâmica molecular, desenvolvido para alto desempenho em simulações de grandes sistemas biomoleculares [KAL99].

O NAMD é distribuído como código livre, e disponibiliza também, um programa gráfico molecular chamado VMD (do inglês, Visual Molecular Dynamics) [TCB08], para configuração da simulação e análise de trajetória e pode ser adquirido de forma independente [NAM08]. NAMD provê escalabilidade para centenas de processadores em plataformas paralelas, dezenas de processadores em clusters de baixo custo e execução individual em PCs. O NAMD é modularizado com linguagem C++ nativa, e é baseado no sistema de programação paralela *Charm++* em conjunto com uma biblioteca específica, onde a computação é decomposta em objetos que interagem pelo envio de mensagens para outros objetos com o mesmo ou com processadores remotos. A estratégia de paralelização é tratada na simulação da molécula por meio de divisões efetuadas no espaço tri-dimensional dos átomos, com cada parte de tamanho suficiente somente para 26 vizinhos mais próximos conservando as partes envolvidas em ligações *bonded* ou *nonbonded* [PHI05]. Assim, esta aplicação emprega o método de decomposição por domínios trazendo alta escalabilidade para as simulações.

2.3.7 Protomol

Protomol é um arcabouço orientado a objetos para DM com licença GPL. Utiliza encapsulamento e programação genérica, com a idéia de prover uma plataforma com extensão de componentes para algoritmos paralelos para DM. Esse programa é inspirado no NAMD2. Porém, seu principal objetivo não é alta escalabilidade, mas fazer uma otimização paralela simples e bastante flexível. Esta abordagem também é encontrada em outros projetos como o POOMA, MTL, Blitz++, entre outros [MAT01].

Este arcabouço aborda o método de decomposição de forças. Possui um esquema de paralelização incremental, que traz suporte a um mecanismo genérico de paralelização e facilidades

para execução em clusters com números moderados de nodos. A comunicação é feita por MPI-2 entre os nodos de processamento do agregado de computadores.

3 TRABALHOS RELACIONADOS

Neste Capítulo será realizada uma discussão de diversos trabalhos relacionados a APIs em aplicações científicas ou de alto desempenho e diferentes formas de se utilizar HPRC com aplicações que envolvem simulações por dinâmica molecular.

3.1 APIs em Aplicações de Alto Desempenho

API é um conjunto de padrões estabelecidos por *software* para utilizar as funcionalidades de aplicativos e não aprofundar no seu entendimento. Uma função amplamente utilizada em plataformas de *hardware* é a adaptação de aplicações para aceleração por *hardware*. Desta forma, utilizam-se os serviços do *hardware* e do *software*, integrando a necessidade da aplicação com os recursos disponíveis no *hardware*.

A função de uma API depende do ambiente onde está, podendo conter vários tipos de funções, dependendo da necessidade. O cuidado no uso de APIs é outro ponto importante, possibilitando tanto acrescentar melhorias, como prejudicar todo o sistema computacional. Existem diversas propostas de APIs na literatura para aplicações de alto desempenho.

O restante desse Capítulo mostra a descrição de uma API e sua utilização de forma correta.

3.1.1 A API OpenFPGA GENAPI

3.1.1.1 Stahlberg et al.

Stahlberg et al. [STA07] propõem uma API genérica (OpenFPGA GENAPI), para dar suporte a integração de programas para cálculos de campos de força clássicos por dinâmica molecular a aceleradores de *hardware* baseados em FPGAs. Dentre as diversas características mostradas como necessidades, apresentam-se inicialmente uma variedade de funções e algumas vantagens na disponibilização de APIs, tais como:

- Alocação de recursos necessários e inicialização do dispositivo FPGA e sua infra-estrutura.
- Gerenciamento dos algoritmos no FPGA (fluxo de arquivos) e seu mapeamento para o dispositivo FPGA.
- Alocação de memória para melhorar a transferência de dados da memória do hospedeiro para os bancos de memória do FPGA e vice-versa.
- Interface explícita para (bloquear) funções de transferência de dados.

Em [STA07], os autores mencionaram melhorias na API adicionando funções específicas para cálculos por dinâmica molecular como:

- Introduzir o conceito de um Algoritmo para registradores e metadados.
- Suporte para codificar e consultar a configuração de *hardware* do FPGA.
- A definição das funções básicas foi dividida em três grupos: Inicialização e Operação, Alocação de memória e Gerenciamento do Algoritmo.

Definiram-se novas características e funcionalidades na API para dar suporte aos cálculos de Lennard-Jones (LJ) e FFT (do inglês, *Fast Fourier Transform*) para interações de longo alcance de Coulomb e a criação da lista de átomos vizinhos. Para isso criaram-se novas funções para uma API portável para DM constituída de 4 categorias: configuração e setup de LJ, API de Operações de LJ, API da camada superior de LJ e baixo nível de configuração FFT.

Na validação da API, em investigações iniciais determinou-se viabilidade nas seguintes plataformas: Mitrion-C, Dime-C, Nallatech H101, Cray XD1 com Virtex-4 LX100 e DRC FPGA. Para examinar o potencial da API na aceleração os autores escolheram uma aplicação por DM, a LAMMPS, para exercitar a API.

3.1.2 A Utilização Correta de APIs

3.1.2.1 Underwood et al.

Underwood et al. [UND06] argumentam que em aplicações HPC há dois aspectos importantes. Primeiro, a forma de uso da API, ocultando o paralelismo do sistema, e as questões relacionadas à arquitetura do sistema como segundo aspecto. Esse trabalho demonstra que uma grande largura de banda, baixa latência na conectividade pode ser importante, mas a maneira correta do uso da API pode ser tão importante quanto. Os autores executaram dois tipos de aplicações, uma FFT e operações DGEMM (do inglês, *Dense Matrix Multiply Operation*), com diversas arquiteturas de *hardware* e diferentes tamanhos de números de ponto flutuante. Além disso, utilizaram-se diferentes API para a conexão entre a máquina hospedeira e a plataforma aceleradora baseada em FPGA. Uma das conclusões do trabalho é que a utilização errada da API pode ter como impacto reduzir em até três vezes o desempenho da aplicação, mostrando a importância da API para a eficiência do sistema.

O desenvolvimento de APIs não é uma tarefa trivial conforme salientam Stalberg et al., e os diferentes ambientes de desenvolvimento fornecidos pelos fabricantes de FPGAs apenas aliviam parte dessa complexidade. Contudo, como FPGAs estão sendo utilizados como a principal técnica na aceleração de *hardware* atualmente, resta o desafio de abordar as diversas plataformas

existentes, implicando um compromisso entre generalidade e eficiência das APIs propostas.

3.1.3 Bibliotecas Portáveis de Núcleos de Hardware

3.1.3.1 Saha et al.

Saha et al. [SAH08] mostram a significativa redução de tempo de desenvolvimento de aplicações com computação reconfigurável, através de bibliotecas portáveis de núcleos de hardware altamente otimizados. Esta literatura apresenta a permuta e os desafios encontrados no projeto de tais bibliotecas, provendo um conjunto de orientações no desenvolvimento de bibliotecas portáveis e a validação desse suporte por meio de um estudo de caso com a biblioteca RCLib (do inglês, Reconfigurable Computing Library). A implementação de núcleos aceleradores de hardware em sistemas HPRC está exposto na Figura 6, onde se tem o núcleo instanciado no wrapper para se adaptar aos sinais providos do núcleo de serviços do próprio fabricante, no qual se cria um alto nível para ser sintetizado e implementado no FPGA. Os autores identificam os 5 principais problemas a serem resolvidos no desenvolvimento desse tipo de biblioteca de hardware, com orientações que levantam importantes características a serem definidas pelo projetista de tais bibliotecas. O primeiro é a análise de domínio, onde determina-se o conteúdo e o escopo da biblioteca, seguido da definição da interface padrão. O terceiro problema é a portabilidade de código, precisa-se considerar um balanceamento entre desempenho e portabilidade. O controle de qualidade é colocado em quarto, e o último é destinado à distribuição, licenciamento e colaboração.

A validação do conjunto de orientações propostos foi realizada por meio de um estudo de caso com a biblioteca RCLib. Esta biblioteca foi desenvolvida pela universidade George Washington (GWU), pela associação acadêmica da universidade de George Mason (GMU) e pela universidade da Carolina do Sul (SC), provê uma coleção de núcleos otimizados de hardware que podem ser portados em diferentes sistemas HPRC, entre esses, estão apenas os principais fabricantes como SGI, SRC e Cray. O conjunto de orientações tem sido completamente validado por meio de estudos de caso com mais de 100 diferentes tipos de núcleos, destinados as aplicações de processamento de imagem, criptografia, bioinformática e aritmética de números inteiros longos. O uso e desempenho da biblioteca de hardware RCLib foi demonstrado por dois exemplos de aplicações, uma utilizando um nodo simples de processamento no algoritmo de redução de dimensão espectral de onda, e o segundo exemplo da RCLib com múltiplos nodos e dois níveis de paralelismo, chip e sistema, em uma aplicação de bioinformática no algoritmo de Smith-Waterman. Nos dois exemplos encontrou-se um nível de desempenho muito maior quando comparado a simples sistemas HPC, na ordem máxima de aproximadamente 32x e 2794x respectivamente em

cada aplicação.

3.2 Aceleração de Processamento com Hardware Dedicado Baseado em FPGAs

O multiprocessamento intra-chip é um tema clássico de pesquisa e desenvolvimento intra-*chip*, visando sempre o alto desempenho como provido por plataformas MPSoC [JER05], populares em aplicações que exigem recursos computacionais avançados, principalmente por sua flexibilidade. Em aplicações de bioinformática e biofísica molecular computacional, exige-se recursos computacionais de alto desempenho e a utilização de plataformas baseadas em FPGA é viável, tendo recebido atenção crescente de diversos autores [AGA07b][GU07][SCR08]. Alguns trabalhos apresentam diversas formas de utilização de plataformas baseadas em FPGAs, como arquiteturas ou máquinas dedicadas [PAT06], escolhas de métodos conforme a aplicação utilizada [CON07] e implementação em aplicações de computação científica [AGA07b] [GU08] [SCR08].

3.2.1 Patel et al.

Patel et al. [PAT06] apontam que mesmo a utilização de um número reduzido de dispositivos FPGAs de baixo custo pode aumentar significativamente o desempenho de tarefas computacionais, em particular aplicações dispendiosas como simulações por dinâmica molecular e de sistemas biológicos. A Figura 8, baseada em Patel et al. [PAT06] mostra uma classificação de organizações de sistemas computacionais paralelos quanto à presença de aceleradores baseados em FPGAs. Como exemplo, a Figura 9 mostra o particionamento de uma aplicação em uma arquitetura proposta pelos mesmos autores para aceleração de simulações por dinâmica molecular, denominada *Toronto Molecular Dynamics* (TMD), uma organização de Classe 3.

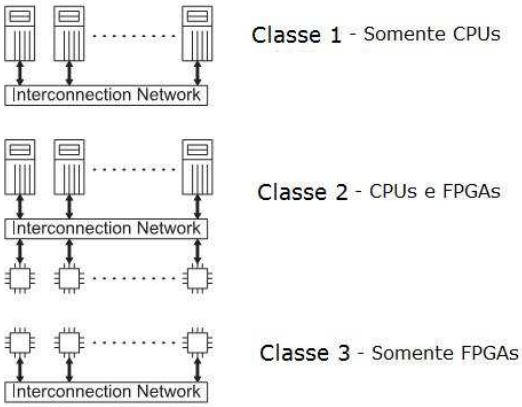


Figura 8 - Uma proposta de classificação de sistemas computacionais paralelos quanto a presença de aceleradores de *hardware* [PAT06].

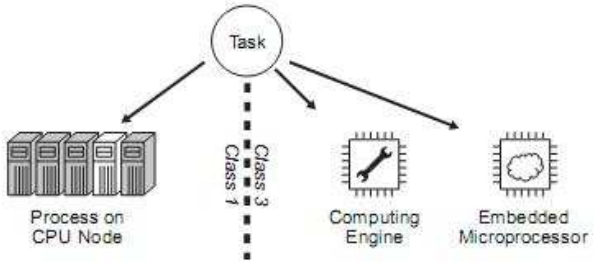


Figura 9 - A execução de tarefas na arquitetura TMD [PAT06].

3.2.2 Alto Desempenho com FPGAs em Bioinformática

3.2.2.1 Conti et al.

Conti et al. [CON07] apresentam diversos métodos relacionados a aceleração por meio de plataformas baseadas em FPGAs, com aplicações em bioinformática e biofísica molecular computacional. Os Autores propõem um método em que se considera a Lei de Amdahl, onde enfatizam a combinação entre a otimização do código original e a atribuição de execução de uma parte desse código ao FPGA, acelerando as tarefas mais exigentes em termos de processamento, e conseqüentemente maximizando o desempenho global da aplicação. Um exemplo de aplicação citada com a utilização desse método são as simulações por dinâmica molecular e aplicações envolvendo a predição de estruturas moleculares com suas interações.

3.2.2.2 Agarwal et al.

Agarwal et al. [AGA07b] demonstram a utilização de FPGAs em computação científica, especificamente em cálculos científicos. Os autores alegam que o uso de HDLs (do inglês, *Hardware Description Languages*) pode ser um fator limitante na área de aceleração de simulações por dinâmica molecular, pois relativamente poucas pessoas da área dominam tais linguagens.

Assim, esses e outros autores sugerem o uso de plataformas reconfiguráveis que contém ambientes de desenvolvimento capazes de gerar *hardware* a partir de linguagens de programação como C. Com tudo, não é abordado em nenhum trabalho da literatura revisada, resultados entre ambientes baseados em HDLs e ambientes baseados em linguagens de programação. Nesta literatura alega-se que, em geral, FPGAs podem prover aceleração de forma satisfatória somente para operações de ponto flutuante com precisão simples. Programou-se uma parte do método PME (do inglês, Particle-Mesh Ewald) que é parte do pacote AMBER, amplamente utilizado para simulação por dinâmica molecular (DM). A Figura 10 mostra a estrutura do bloco de controle e do bloco de dados da estrutura implementada em uma plataforma baseada em FPGAs, bem como a comunicação entre o hospedeiro e a plataforma. O gerenciamento da comunicação entre a máquina hospedeira e a plataforma é realizado pela ferramenta MAP, que se ocupa da transferência de dados, da alocação de memória, e da distribuição de tarefas para o FPGA de controle do FPGA, como pode ser visto na Figura 10. A implementação foi realizada com o *software* AMBER e a plataforma SRC-6E *Mapstation*, com moléculas de 23.558 e 61.641 átomos. Os autores mostraram um desempenho de 3.30x e 3.97x respectivamente ao tamanho da molécula, realizando uma otimização no gerenciamento da memória.

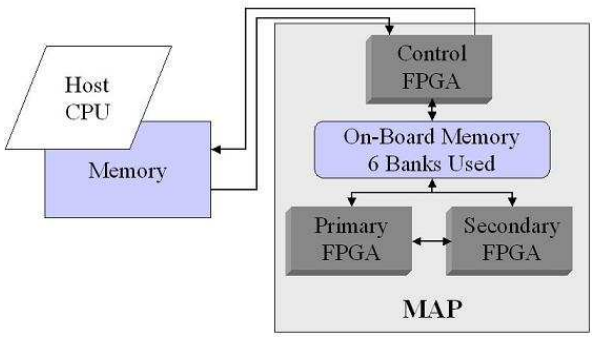


Figura 10 - Controle e data path entre o host e os dispositivos FPGAs [AGA07a].

Os Autores de [CON07] mostram uma diversidade de métodos para execuções de aplicações HPC em FPGA, principalmente em áreas relacionadas a bioinformática. Já os estudos de [AGA07b], além de sua contribuição conceitual, agregam dados de desempenho para aplicações envolvendo cálculos para simulações por dinâmica molecular. A implementação foi realizada especificamente no cálculo de PME utilizando diferentes análises e arquiteturas.

3.2.2.3 Scrofano et al.

Scrofano et al. [SCR08] empregam aceleradores por *hardware* baseados em FPGAs em aplicações envolvendo simulações por dinâmica molecular. Nesta implementação toda a aplicação de simulação por dinâmica molecular foi implementada, os resultados contém o desempenho total

das simulações. Foram comparados dois ambientes, com e sem *hardware* reconfigurável (HR), de 1 a 8 nodos de processamento. Cada nodo com HR representa uma ou mais CPUs e um FPGA, similar a classe 2 da Figura 8.

Cálculo de forças entre átomos não ligados consome grande parte do tempo de execução da aplicação, por isso, foram convertidos para *hardware* com o *software The Carter Compiler* presente na plataforma. Utilizaram as plataformas *MAPstation* SRC-6 series-C e series-E, em cada elemento de *hardware* reconfigurável tem-se Virtex-II XC2V6000 e XC2VP100s respectivamente, e todos FPGAs operando a 100 MHz.

O *software* por DM foi escrito pelos autores para programar o algoritmo *velocity Verlet* em simples e dupla precisão aritmética. Em átomos não ligados, foram efetuados cálculos de Lennard-Jones e Coulomb para as forças e o movimento dos átomos, em duas moléculas o ácido Palmítico de 52.000 de átomos e a proteína CheY de 32.000 de átomos. Nos resultados dos experimentos as duas moléculas obtiveram um aumento de 2x com aceleração por *hardware*. Porém, na implementação com vários nodos obteve-se resultados inferiores, quando a quantidade de nodos é maior que um.

É importante ressaltar que, houveram diferenças, na análise das saídas a flutuação de energia nas simulações aceleradas ou não, principalmente na molécula do ácido Palmítico, que aproximou-se de 5% continuamente em toda simulação. A proteína CheY aproximou-se de 5% apenas em um momento no início da simulação, já no restante da simulação encontrou-se similar ao original (não acelerado).

3.2.2.4 Gu et al.

Gu et al. [GU08] apresentam aceleração por intermédio de co-processadores baseados em FPGAs para forças de curto alcance. Foi realizada uma otimização no código do cálculo de Lennard-Jones (curto alcance), que inclui os métodos de atração de Van Der Waals e repulsão de Pauli.

Uma das principais características otimizada está relacionada a interpolação aplicada na computação de forças de curto alcance. Compararam-se três métodos de interpolação de Taylor, Polinômios Ortogonais e Hermite, levando em consideração a quantidade de seções e a ordem da interpolação. Escolheu-se o método Ortogonal, devido principalmente a redução de erros, e caracterizou-o para executar o pipeline de forma eficiente. Outras duas características importantes na otimização foram a precisão nos cálculos com ponto flutuante e a exclusão de átomos ligados, antes de efetuar os cálculos de átomos não ligados. No primeiro comparou-se dois modelos com ponto flutuante completo (precisão dupla) e um semi-ponto flutuante (com 35 bits). Na outra

característica, diminuiu-se o número de átomos e conseqüentemente a computação efetuada nesses cálculos.

Na definição do projeto do co-processador o uso de *pipelines* foi essencial na caracterização do *hardware* para a aceleração. No cálculo de forças também foi utilizado no suporte ao método de listas de células e o gerenciamento de memória, pela quantidade de informações transferidas nesses cálculos.

Os experimentos foram realizados com uma molécula de 77 k de átomos e o seu PDB ID é 1qgk. A caixa que envolve esta molécula possui dimensões de 93 Å x 93 Å x 93 Å, a execução da simulação para esta molécula é de 1.000 passos de tempo. Utilizaram o *software* Protomol para simular a molécula e no código base, as otimizações em ponto flutuante foram de 35 bits com 4 *pipelines* no cálculo de forças.

Compararam-se duas versões, acelerada e não acelerada, e com dois *softwares* o Protomol e o NAMD. No Protomol conseguiu-se um aumento de 9.8x, já com o NAMD esse número caiu para 8.8x. Os autores realizaram outros experimentos com mais detalhes na configuração do acelerador FPGA, obtendo-se com dois dispositivos XC2VP70 (da família VirtexII Pro da Xilinx) um ganho de 11.0x com o NAMD realizado no laboratório próprio e de 6.5x com o mesmo *software* em um laboratório externo, da mesma forma, com um VP100 obteve-se 8.9x e 5.3x respectivamente e para a nova Virtex-5 LX330T com precisão simples o aumento foi de 16.8 e 10x conforme os parâmetros de *software* anteriores.

3.3 Comparação das abordagens

Existem diferentes grupos de pesquisa trabalhando com o objetivo de ajudar na aceleração de aplicações por DM aprimorando o desempenho em arquiteturas modernas de computadores baseadas em *hardware* reconfigurável, como [AGA07b] [GU08] [KIN06] [PAT06] e [SCR08]. Em biofísica molecular, há a necessidade de alto poder computacional para facilitar a obtenção de dados e posterior análise do sistema molecular por meio de simulações. Se esta área encontrasse a quantidade de informações necessárias para analisar o sistema molecular em segundos, mesmo em grandes sistemas ou tempos de simulação, seria possível encontrar fármacos para doenças em poucas horas, constituindo um sonho aos pesquisadores de biofísica molecular.

Pode-se observar na Seção 3.2, que mesmo com a diversidade dos autores trabalhando com aceleração por *hardware* em uma parte dos cálculos (curta distância) de toda a dinâmica molecular, ainda existe um campo amplo para pesquisa, por envolver uma infinidade de abordagens como: métodos, técnicas, algoritmos, linguagens de programação, bibliotecas, plataformas FPGAs, aplicações por DM, entre outros. Na Tabela 2, apresenta-se de maneira simplificada os principais

pontos nas acelerações baseadas em *hardware* reconfigurável para aplicações por DM, como: algoritmo paralelo acelerado, aceleração obtida, plataforma utilizada, otimizações, tamanho da molécula e compilador para HDL. Nestas literaturas, a que possui maior similaridade com o problema a ser tratado por esse trabalho é a de Agarwal et al. Porém, existem diferenças explícitas entre as abordagens, em principal, na otimização do algoritmo acelerado, na plataforma utilizada e a API, são as principais. Não será utilizada uma API comercial como muitos autores o fizeram. A proposta desse trabalho é construir uma API para comunicação entre a máquina hospedeira e a plataforma FPGA.

Tabela 2 – Revisão do estado da arte em aceleração de aplicações de simulação por dinâmica molecular usando hardware reconfigurável.

Autores	**Algoritmo Paralelo Acelerado**	**Aceleração Obtida**	**Plataforma Utilizada**	**Otimização**	**Tamanho da Molécula**	**Aplicação por DM**	**Compilador p/ HDL**
Patel et al.[06]	Simulação por DM Completa	N.A.	TMD - XC2VP100	Organização de Hardware	N.A.	NAMD	N.A.
Agarwal et al. [07]	PME direto	3.30x e 3.97x	Mapstation SRC-6E	Memória	23.558 e 61.641	AMBER 8	The Carter Compiler
Scrofano et al.[08]	Velocidade de Verlet	Nodo único 2.0x , c/ Múltiplos não obteve	Mapstation SRC-6 C e E	Memória e precisão de PF	32.000 e 52.000	Própria	The Carter Compiler
Gu et al. [08]	Lennard-Jones	VP70s(PD)-11.0x VP100(PD)-8.9x LX330T(PS)-16.8x	WILDSTAR II Pro	Interpolação, precisão de PF e exclusão de átomos	77.000	Protomol e NAMD2	N.A.
Proposta desse Trabalho	PME direto	N.A.	Virtex 4 - FX100	Parte do cálculo de curto alcance	35.681	AMBER 9 (PMEMD)	Impulse C

4 MATERIAIS E MÉTODOS

Neste Capítulo serão descritos todos os recursos utilizados nos experimentos práticos realizados nesse trabalho. Algumas características específicas de cada experimento serão descritas na própria Seção que define o experimento para facilitar a compreensão.

4.1 Recursos de Software

Os estudos de casos realizados nos Capítulos 6, 7 e Seção 7.1, foram realizados com o mesmo pacote de aplicações por dinâmica molecular, o AMBER [AMB09] na versão 9. Nos estudos de casos escolheram-se dois subconjuntos desse pacote, o SANDER e o PMEMD como os *softwares* para aplicação por dinâmica molecular, e no da Seção 7.1 apenas o PMEMD. Esses experimentos foram executados no laboratório do GAPH [GAP08].

Nesses estudos de caso empregou-se o sistema operacional Linux, baseado em software livre, com a distribuição Fedora. Nos experimentos do Capítulo 5 fez-se uso da versão 9 com a arquitetura X86_64 nas duas máquinas. Devido às restrições de *hardware* e *software*, que serão apresentados no Capítulo 6, os testes realizados com a plataforma FPGA e a API, na Seção 7.1, aplicou-se na máquina hospedeira o sistema operacional Linux Fedora na versão 8 e arquitetura i386.

Para a compilação dos softwares do pacote AMBER foi necessário a utilização de compiladores do projeto GNU e outros específicos para as linguagens Fortran e C. Os compiladores GNU foram o GFORTRAN e GCC nas versões 4.1 e 4.3, e do fabricante Intel, o IFORT na versão 10.1, sem a biblioteca MKL (do inglês, Math Kernel Library) para cálculos matemáticos e o compilador ICC.

Na comunicação por meio de troca de mensagens, nos experimentos que utilizam o modo paralelo para realizar as simulações por DM, empregou-se a biblioteca MPICH na versão 2.

Com o *software* VMD [TCB08], na versão 1.8.6, realizou-se a animação e visualização em 3D do sistema molecular nas Figuras 12 e 13. Este software possibilita a definição de cores e modos de visualização conforme descritos nessas figuras.

4.2 Recursos de Hardware

Para os estudos de casos utilizaram-se dois ambientes, o primeiro, referente às Seções 5.1 e 5.2, realizou-se por intermédio de duas máquinas interligadas, em uma rede *Fast-Ethernet*, organizadas em um pequeno *cluster*. O segundo ambiente, referente a Seção 7.1, contém apenas

uma máquina reconfigurável com a plataforma de *hardware* (FPGA) acoplada ao barramento PCI.

As duas máquinas definidas para o primeiro ambiente possuem as seguintes características:

1. Processador Intel Core 2 Quad de 2.40 GHz e 4 GB de memória RAM
2. Processador AMD Turion x 2 de 1.6 GHz e 1 GB de memória RAM

No segundo ambiente a configuração básica do computador, máquina hospedeira, e da plataforma de hardware são, respectivamente:

1. Processador Intel(R) Pentium(R) 4 CPU 2.40GHz e 1 GB de memória RAM
2. Placa DN8000K10PCI com um FPGA, o XCV4FX100 da Xilinx, descrita na Seção 6.1.1.

4.3 Sistema Molecular

O átomo é a unidade básica da matéria, são os responsáveis em compor os seres vivos ou minerais, possuem um núcleo envolvido por uma nuvem de cargas elétricas negativas. As moléculas são formadas por ligações dentre dois ou mais átomos (N > 1). Um sistema molecular é formado por um conjunto de átomos e/ou moléculas, e podem conter milhares ou milhões de átomos dependendo do seu tamanho. Os parâmetros iniciais para a simulação por dinâmica molecular e o sistema molecular, definidos nos arquivos de entrada nos experimentos, foram obtidos do grupo de pesquisa LABIO [LAB08]. Estas características são iguais em todos os experimentos apresentados posteriormente, o sistema molecular utilizado está exposto na Figura 11.

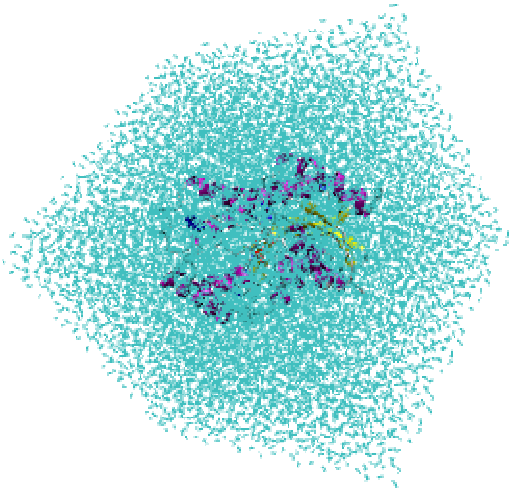


Figura 11 – O sistema molecular é composto por uma molécula de proteína, uma coenzima e 6 contra-íons. Esta molécula é imersa em uma caixa ortorrômbica de dimensão 72,372 Å x 68,278 Å x 72,019 Å preenchida por moléculas de água. Esse sistema molecular contém 35.681 átomos e 10.532 moléculas de água e a molécula possui o PDB ID 1ENY. Foram aplicados estilos de desenho distintos da seguinte forma: na estrutura secundária da molécula é novo Cartoon com coloração do tipo estrutura, nas moléculas de água é linhas com coloração elemento e no resíduo NAH é com coloração do tipo palito.

A dinâmica molecular utilizada na simulação é composta por pressão de 1 atm, temperatura de 298,16 Kelvins ou 25° C, caracterizando no conjunto ou *ensemble* isobárico-isotérmico (NPT), e nos cálculos de átomos não ligados utiliza-se o PME, van der Waals, entre outros, expostos na Seção 5.2. A molécula possui 20 tipos diferentes de átomos e todo o sistema com a caixa ortorrômbica é apresentada na Figura 11, esta imagem foi criada a partir do *software* VMD, descrito na Seção 2.3.6.

A estrutura terciária, mostrada na Figura 11, faz parte de um experimento já iniciado pelo grupo LABIO [LAB08]. Esta imagem foi retirada do intervalo de 9.900 ps a 9.910 ps da simulação desse sistema molecular, os parâmetros da simulação estão descritos na Seção 5.2. A estrutura terciária desse sistema molecular pode ser visualizada de forma mais detalhada sem a caixa ortorrômbica e o solvente na Figura 12. Esta figura foi criada com o *software* VMD determinando cada característica das estruturas por meio de diferentes métodos de coloração e de desenho para melhor visualização da estrutura terciária. A animação foi representada em 3D pelo *software* VMD com o método de coloração estrutura, com material brilhoso e método de desenho novo cartoon.

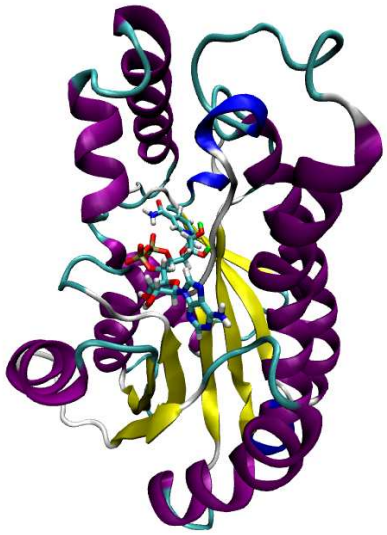


Figura 12 – Estrutura Terciária da molécula sem a caixa ortorrômbica, e caracterizando o resíduo NAH pelo desenho em forma de palitos no centro da estrutura da molécula.

4.4 Dinâmica Molecular

Os parâmetros definidos na dinâmica molecular utilizada está contido no arquivo de entrada da simulação do sistema molecular. Esses parâmetros foram definidos previamente, conforme as características estipuladas para a simulação, como os parâmetros do conjunto estático utilizado, o uso do método PME, entre outros. A descrição e definição de todos os parâmetros que compõem esse arquivo de entrada estão presentes no Anexo A – Características da Dinâmica Molecular. Este

exemplo, é utilizado apenas para o pacote AMBER, o arquivo apresentado no anexo é para ser utilizado no SANDER. Para utilizar a mesma dinamica molecular no PMEMD é preciso retirar os parâmetros **`bellymask`**, **`restraint_wt`**, **`restraintmask`**, **`isgsta`**, **`isgend`** e **`iamoeba`**, que não são suportados por essa aplicação.

5 ESTUDOS DE CASOS

Dois estudos de caso são realizados neste Capítulo, um para a investigação da aplicação PMEMD e outro para realizar simulações por dinâmica molecular. Na última Seção é feita uma discussão sobre a biblioteca MPI e sua possível utilização.

5.1 Investigando o Perfil da Aplicação PMEMD

Recentemente, observou-se que os níveis de complexidade nas aplicações aumentam de maneira espantosa, especialmente em aplicações científicas. Além disso, o grande número de módulos, rotinas e funções acoplados dificulta a compreensão do problema, nestas aplicações, que geralmente só é entendido pelos próprios autores.

Para auxiliar o entendimento de aplicações complexas têm-se ferramentas que atuam como monitores de traçado de perfil da mesma, coletando informações específicas e essenciais para sua análise. É possível obter informações a respeito do tempo de execução em cada módulo ou rotina, possibilitando uma análise de qual parte do código exige maior poder computacional, e coletar a quantidade de chamadas efetuadas de cada módulo ou rotina da aplicação. Estas informações facilitam o trabalho de paralelização de uma aplicação que necessita de HPC.

Como visto na Seção 2.3, aplicações de simulação por dinâmica molecular necessitam de HPC e contêm um grande nível de complexidade. Para facilitar o entendimento do problema, utilizou-se uma ferramenta para traçar o perfil do *software* PMEMD. Esta ferramenta é chamada de GPROF (do inglês, *GNU PROFiler*) [GRA82], que possui licença GNU, e geralmente é um componente básico de qualquer distribuição, em sistemas operacionais Linux.

Para compreender a aplicação PMEMD de maneira mais profunda, adquiriram-se informações por intermédio de duas técnicas empregadas, o plano de perfil e o gráfico de chamadas, obtidas pela ferramenta de *profiling*. No plano de perfil visualizam-se com detalhes os tempos gastos em cada módulo e rotina da aplicação, conforme Tabela 3. Já no gráfico de chamadas além da quantidade de chamadas em cada módulo ou rotina, obtém-se a hierarquia e seqüência das chamadas efetuadas em toda aplicação. Esses resultados estão expostos na Figura 13, a quantidade de chamadas em cada rotina está entre colchetes e os números em cada rotina mostra a sequência de execução dessas no *software* PMEMD. Na sub-rotina short_ene, pode-se observar a estrutura básica desta parte do *software*, com seus laços e seleções, e a quantidade de *loops* feitos em cada seleção, representado pelos números entre colchetes.

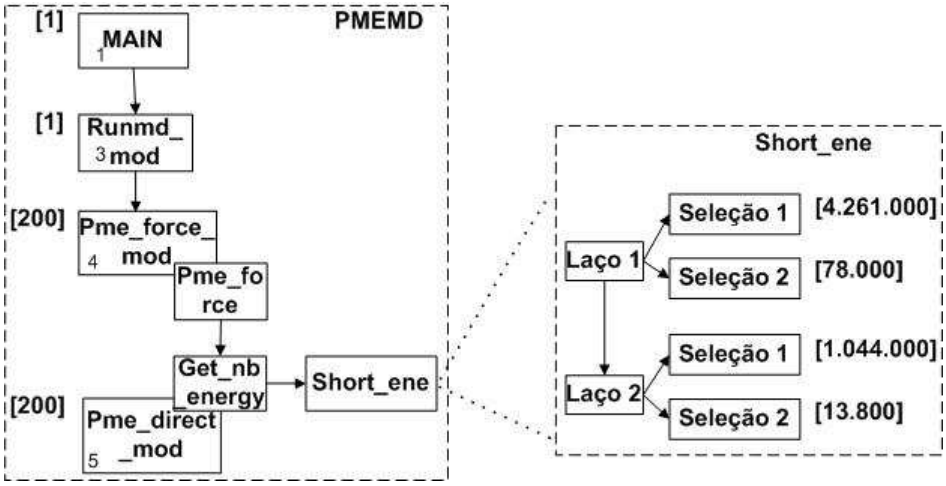


Figura 13 – Gráfico de Chamadas de uma simulação por dinâmica molecular com 400 fs utilizando o PMEMD. Os campos definidos com colchetes representam a quantidade de chamadas realizadas na rotina, módulo ou trecho de código pela ferramenta GPROF. O lado esquerdo foi adquirido por meio de testes de monitoramento inseridos no código da aplicação PMEMD com os mesmos parâmetros de entrada.

5.1.1 Investigação do Perfil

Para caracterizar de maneira mais detalhada alguns pontos principais em uma aplicação é realizado alguns procedimentos para poder traçar o perfil do *software*. Na intenção de trabalhar em uma parte da aplicação em que se tenha um custo computacional alto, faz-se necessária a investigação do perfil da aplicação, capturando informações de tempos e quantidades de chamadas em cada rotina do programa.

Na aplicação de Simulação por dinâmica molecular PMEMD, *software* usado nesse trabalho, são executados 5 módulos, conforme sequência da Figura 13, até se encontrar a sub-rotina *short_ene*. Essa rotina interna usada para a realização desse trabalho por conter o maior custo computacional em toda simulação, conforme Tabela 3. Nesta se mostra a rotina *get_nb_energy* que efetua a chamada a *short_ene*, e que faz o cálculo de forças dos átomos não ligados.

Os valores mostrados nas Figura 13 e na Tabela 3, foram adquiridos de uma simulação utilizando o PMEMD com 200 *time-steps* e com uma taxa de 0,002 ps para cada passo (*step*). Foi estipulado este valor pela quantidade de dados gerados no monitoramento da subrotina *short_ene*, tornando-se difícil o acesso aos dados pelo seu tamanho. Com isso, para que os tempos de simulação não contenham medidas diferentes no monitoramento e no traçado de perfil, optou-se por este número, que mostra claramente quais as funções mais visitadas, a hierarquia e os tempos gastos iguais ao de uma simulação maior. Estas informações têm alta relevância ao estudo desta aplicação.

Tabela 3 – Traçado de Perfil do PMEMD com 200 passos por simulação com a ferramenta **gprof**.

Módulo (Rotina(Rot. Interna))	Tempo (%)	Tempo (seg)
pme_direct(get_nb_energy)	64.32	214.65
nb_pairlist(get_nb_list)	14.13	47.14
pme_recip(do_pmesh_kspace)	4.98	16.61
pme_recip(fill_charge_grid)	1.70	5.68
pme_fft(fft3drc_back)	1.66	5.53
pme_force(pme_force)	1.49	4.97
pme_fft(fft3drc_forward)	1.40	4.68
fft1d(cfftf1(passf3))	1.10	3.66

Cada chamada a rotina *get_nb_energy* gera quantidades diferentes de repetições nos laços dentro da subrotina *short_ene*. Com isso, foi realizado um cálculo simples de média aritmética na quantidade de repetições de algumas amostras (ou chamadas a rotina *get_nb_energy*), determinando os valores expostos na Figura 13, nas chamadas efetuadas dentro da rotina interna *short_ene_novec*. Os resultados obtidos em toda execução do PMEMD (à esquerda da Figura 13) foram obtidas, pela ferramenta **gprof**.

Com esses resultados, obtidos por meio da ferramenta **gprof**, foi possível determinar o trecho de código ideal para efetuar a paralelização e conseqüentemente o transporte desse código para plataforma de *hardware* para efetuar o objetivo geral de aceleração da aplicação de simulação de moléculas. Em [KUE05], mostra-se esta mesma rotina (*Short_ene*) com maior custo computacional para simulações utilizando o SANDER, com 73.14% do tempo total dos cálculos de forças em átomos não ligados que chegam a 70.82% do tempo de execução total da aplicação. Na Figura 14 pode-se observar a porcentagem de cada cálculo com o JAC (do inglês, Joint Amber-Charmm) *benchmark*, que está presente no pacote do AMBER.

No SANDER foram definidos os mesmos parâmetros de entrada apresentados pela dinâmica molecular, mas com 5.000 passos de tempo ou 10 ps de tempo de simulação, utilizando a máquina com o Turion x 2, executando em modo serial. Como discutido anteriormente o cálculo na rotina *Short_ene* é o que contém maior custo computacional em toda simulação, isto pode ser comprovado pelo JAC *benchmark* da Figura 14. Estes valores foram adquiridos do arquivo de saída gerado pelo JAC e reduzido a quantidade de informações para facilitar a visualização. Cada rotina vem associada com segundos consumida pela sua execução em toda a simulação e organizada hierarquicamente. A rotina *Short_ene* gastou 4.863,15 segundos, de um total de 7.071,21 segundos, chegando a aproximadamente 68,77% de todo o tempo de simulação, comprovando assim, que trata-se da rotina que possui maior custo computacional.

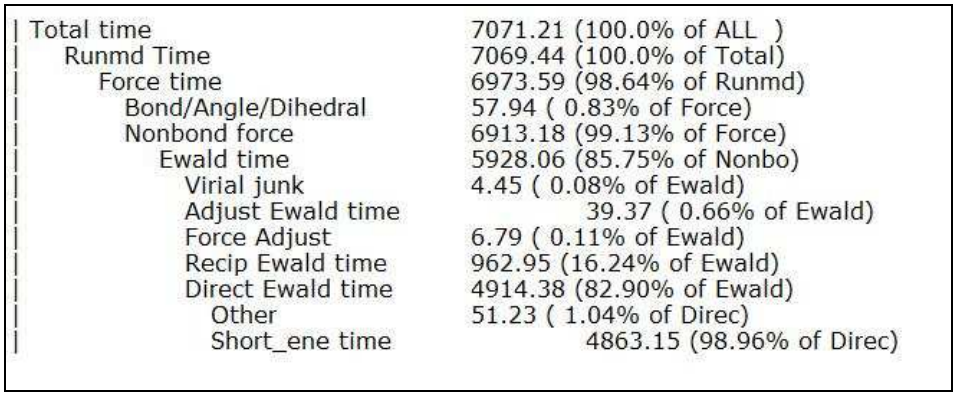


Figura 14 - Porcentagem dos Cálculos em Simulações por DM realizado pelo JAC Benchmark do AMBER 9. Com o tempo de execução das principais rotinas ou módulos existentes em segundos e sua respectiva porcentagem.

5.2 Realização de Simulações por Dinâmica Molecular com AMBER

Para o entendimento de todo o processo de simulação é necessária a compreensão de áreas co-relacionadas como a física, química e biologia. Existem grandes quantidades de cálculos realizados que não são visíveis ao usuário da aplicação. Como mostrado na Seção 2.2, existe uma grande variedade de aplicações que utilizam dinâmica molecular para simular sistemas biológicos de vários tamanhos.

5.2.1 Experimentos

Na Instalação do pacote AMBER deve-se escolher entre uma instalação serial ou paralela. Na serial a sua execução é realizada em apenas um núcleo de processamento, mesmo que o processador contenha mais núcleos. Já na paralela, pode-se definir a quantidade de processos iniciados. Se o número de processos for maior que a quantidade de núcleos é realizado um particionamento entre eles e o escalonamento entre todos é realizado pelo sistema operacional. Esta é uma técnica baseada em concorrência entre processos. É possível também utilizar um cluster de máquinas e distribuir a quantidade de processos para cada nodo do *cluster*, conforme se desejar.

A Simulação realizada com PMEMD contém a mesmas características de entrada da dinâmica molecular apresentadas na Capítulo 2, e o tempo de simulação foi definido como 10 ps ou 5.000 *time-steps*.

Os gráficos desta Seção apresentam os resultados das duas aplicações por DM (PMEMD e SANDER) em diferentes ambientes, para efeito de comparação entre estas. Os recursos de hardware possuem as configurações apresentadas na Seção 4.2. Os resultados foram obtidos com o modo paralelo, por intermédio de um cluster empregando-se a comunicação dos nodos por meio da biblioteca MPI. Essa comunicação também foi utilizada nos testes com mais de um processo na

mesma máquina quando executado independentemente. As simulações usaram 8 diferentes quantidades de processos: 1, 2, 3, 4, 5, 6, 8 e 12. Na versão paralela não foi realizada simulação com 1 processo ou serial por motivos óbvios.

A Figura 15 mostra os resultados para as duas máquinas independentes, com simulações utilizando as duas aplicações por DM. A diferença relacionada entre os dois processadores pode ser observada claramente em qualquer uma das duas aplicações, em que o tempo de simulação é muito maior utilizando um processador de dois núcleos em qualquer situação.

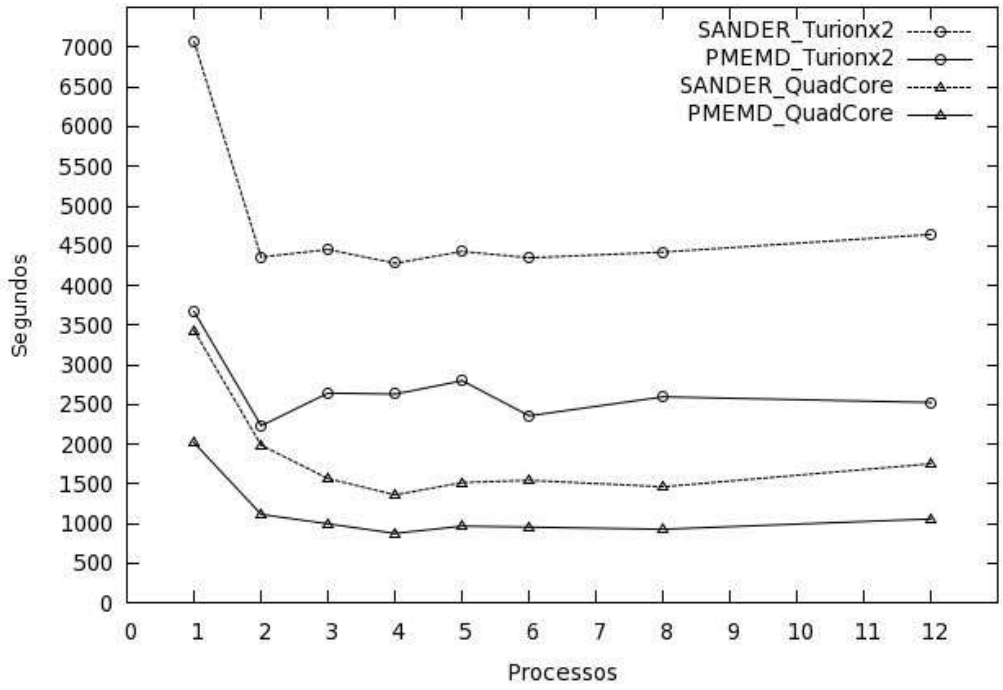


Figura 15 – Simulação por DM utilizando PMEMD e o SANDER em duas máquinas com processadores distintos (Intel QuadCore e AMD Turion x 2).

Nas simulações com o ambiente paralelo, cluster ou agregado de computadores, obteve-se resultados ruins em relação aos resultados observados com apenas uma máquina. Esses resultados aparecem na Figura 16 e na Figura 17, mostrando que mesmo com um agregado de computadores, o tempo de simulação para o mesmo sistema, é maior do que com um computador possuindo um processador de quatro núcleos, mostrando a possível má influência da interface de comunicação na simulação.

Conforme os gráficos apresentados anteriormente, nesta Seção pode-se observar que mesmo em um ambiente paralelo não se conseguiu obter um desempenho melhor do que em uma máquina com um processador com vários núcleos de processamento, mostrando a possível má influência da interface de comunicação na simulação. O ambiente paralelo mostrado foi implementado com uma interface de rede *Fast-Ethernet* de 100 Mbps para conexão entre os processadores, e a comunicação é realizada pela biblioteca MPI, resultando em aproximadamente 12 GB de transferência de dados

para cada simulação, mostrados por meio de um monitor do sistema do próprio sistema operacional.

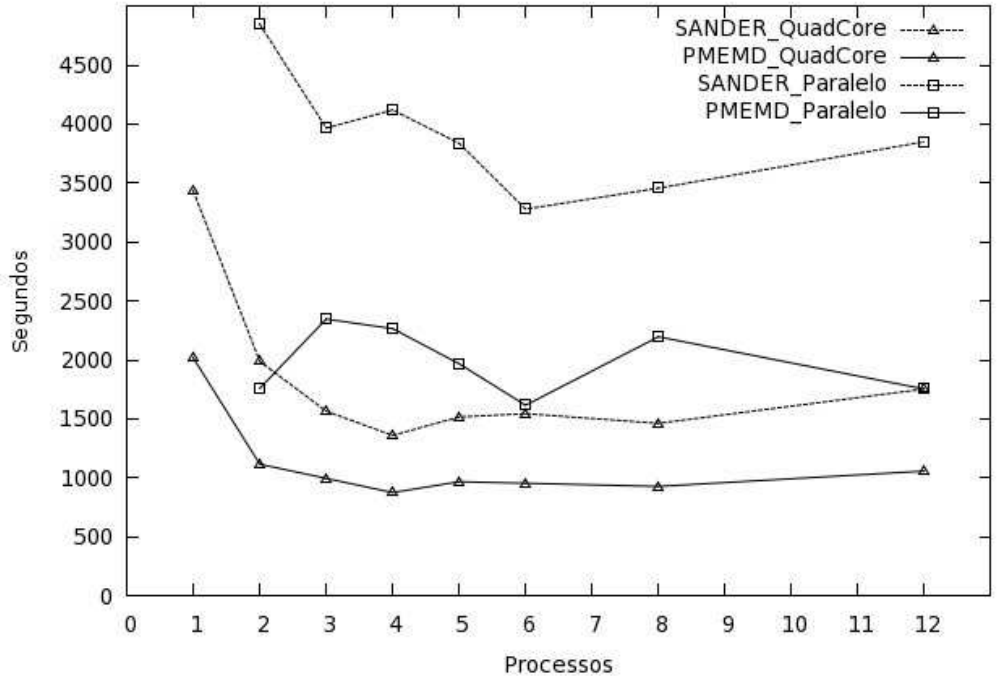


Figura 16 – Simulação por DM com o PMEMD e o SANDER, em um ambiente paralelo e uma máquina com processador Intel QuadCore.

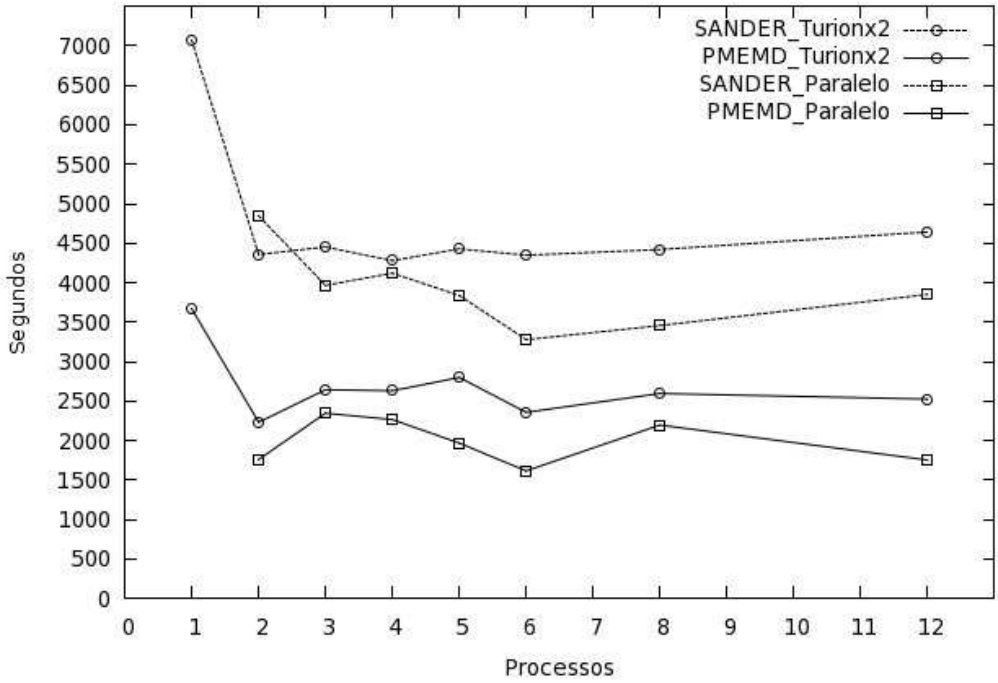


Figura 17 - Simulação por DM com o PMEMD e o SANDER, em um ambiente paralelo e uma máquina com processador AMD Turion x 2.

Outro ponto importante a ser observado e o principal estímulo para a realização destas simulações é a comparação entre as duas aplicações pertencentes ao pacote AMBER, PMEMD e SANDER. Em todos os ambientes mostrados a aplicação PMEMD foi superior ao SANDER,

mesmo com arquiteturas e quantidades de processos distintos. Por esta razão, e pela simplicidade e otimização do código do PMEMD, definiu-se que esta será a aplicação base para o trabalho a ser realizado.

5.3 Comunicação com MPI

Processamento paralelo permite o aumento de desempenho de um programa utilizando técnicas para aproveitar melhor os recursos de *hardware* de um sistema. A comunicação em processamento paralelo usando agregados ou computadores paralelos, permite a agregação de técnicas de sistemas distribuídos para a troca de informações entre processadores do sistema, como a troca de mensagens. Em processamento paralelo definem-se entre os computadores pertencentes ao agregado dois tipos, o mestre que realiza a distribuição e gerenciamento das tarefas e o escravo que realiza a computação das tarefas. A comunicação pode ser efetuada entre mestre e escravos ou entre escravos e escravos. Dependendo da aplicação a ser paralelizada, utiliza-se um desses tipos ou os dois em conjunto. Para efetuar essa comunicação exige-se um nível de complexidade alto em programação paralela e distribuída. Vários fatores envolvidos em eventos de comunicação devem ser coordenados, tais como a transferência segura de arquivos, a identificação e abertura de portas nas máquinas participantes, o particionamento dos dados a serem transmitidos, o controle de chegada e de envio dos dados, padrões para efetuar solicitações, entre outros.

Uma biblioteca popular para a comunicação com estas funções é a MPI [GRO99], que implementa técnicas de troca de mensagens em processamento paralelo. Existem diversos programas e bibliotecas que efetuam as técnicas de troca de mensagens por meio de primitivas MPI, os principais são: MPICH [GRO96], LAM/MPI [BUR94] e OpenMPI [GRA05].

A biblioteca utilizada para efetuar as simulações da Seção 5.2 foi a MPICH [GRO96] . Observou-se nestas implementações que esta biblioteca, além das primitivas padronizadas da MPI, encontra-se um programa chamado MPD (do inglês, Management Processor Daemon). Este faz a divisão e cria o anel base para a comunicação dos processos entre as máquinas. Este gerenciador de processadores precisa ser iniciado antes da execução das primitivas MPI, utilizando uma porta informada ao cliente e iniciada no servidor para poder ser adicionado no anel. É requisitada a configuração de IP associado ao nome da máquina e um protocolo para autenticação remota, podendo-se escolher entre ssh (padrão) e rsh. Além disso, a biblioteca contém uma dependência no local onde se encontra o *software* a ser paralelizado, por exemplo, se o programa executável está no caminho "/home/user/programa" no servidor, ele obrigatoriamente tem que estar em todas as máquinas pertencentes ao anel nesse mesmo local. Esta dependência acontece na biblioteca MPI devido à utilização de técnicas de processamento paralelo baseado em SPMD (do inglês, *Single*

Program, Multiple Data).

Na execução paralela utilizando MPI com aplicações da Seção 5.2, observou-se uma quantidade grande de memória utilizada em cada processo iniciado na máquina, além do processo mestre que geralmente é ainda maior. A partir de dados capturados com o monitor do sistema operacional, verificou-se que o PMEMD utiliza em cada processo escravo iniciado aproximadamente 35 MB, e no SANDER esse valor sobe para 45MB, dados capturados com monitor do sistema do sistema operacional. Com isso, a transferência de dados entre as máquinas pode influenciar no desempenho de todo o sistema como averiguado na Seção 5.2. Com a necessidade de memórias de grande porte, requisitos reservados de memória nas plataformas FPGA, podendo inviabilizar implementação em aceleradores por *hardware*. Em [SAL06], explora a sobrecarga causada por requisitos pesados de memória em *hardware* reconfigurável.

Os autores de [SAL06], relatam que a utilização do padrão MPI necessita de vários recursos que podem não estar disponíveis em sistemas embarcados ou plataformas FPGA. Isto inclui sistema operacional, alta capacidade de memória, processador, entre outros. Os autores implementaram um *subset* da MPI denominado TMD que não utiliza sistema operacional e possui uma necessidade menor no tamanho da memória (8.7 KB), mas esta implementação é específica para máquinas TMD. Para [WIL06], o ambiente ideal para MPI é aplicado ao projeto de SOCs [MAR01] complexos inclusive com uso de NoCs [BEN02], já que pode-se demonstrar que barramentos compartilhados em MPSoCs não são escaláveis além de 8 processadores.

Como uma parte da aplicação por DM é enviada para os nodos clientes por meio de MPI, a implementação em FPGAs seria comprometida, tendo que explorar a utilização de processadores nos FPGAs. Assim, o emprego de um nível de paralelismo de grão pequeno não seria possível. Para conseguir enviar só os dados necessários pelas primitivas MPI, reduzindo a quantidade de chamadas e de transferência de dados teria-se que verificar e modificar grande parte do código da aplicação. Assim, torna-se o trabalho mais complexo, conforme Seção 5.1, limitando o nível de paralelismo que se pode obter com FPGAs.

Devido à quantidade de restrições impostas na implementação da aplicação por DM utilizando a biblioteca MPI decidiu-se que a comunicação entre a máquina hospedeira e a plataforma FPGA não será realizada por intermédio de primitivas MPI.

6 IMPLEMENTAÇÃO

6.1 Recursos Utilizados

Esta Seção apresenta os recursos de *hardware* e *software* utilizados para criar e testar a API proposta. Estes recursos são apresentados de forma sucinta junto às dificuldades para seu uso.

6.1.1 Plataforma de Hardware (FPGA)

Uma plataforma muito difundida na aceleração por *hardware* é o FPGA. Também chamado de *hardware* reconfigurável, estas plataformas contêm algumas vantagens e desvantagens no seu emprego. As vantagens estão no custo, na facilidade de alterações no *hardware* e o nível de paralelismo possibilitado. Como desvantagens pode-se citar o alto consumo de energia e a relativa escassez de recursos dos chips dificultando a criação de grandes códigos, e a dificuldade de encontrar projetistas de hardware. A placa destinada a realização dos testes iniciais desse trabalho foi a DN8000K10PCI do Grupo DINI [DIN09]. Esta placa possui características interessantes para o trabalho, a mais importante destas é a utilização da interface PCI para comunicação com a máquina hospedeira e a possibilidade de inserção de até 3 FPGAs na mesma placa, como pode ser observado na Figura 18 o diagrama de blocos caracterizando esta plataforma FPGA.

Para validação da interface PCI, dois componentes possuem grande importância para a comunicação. O primeiro destes é o CI controlador QL5064 que interage diretamente com o slot PCI e o segundo é um FPGA Spartan II para interconexão e configuração do(s) FPGA(s) destino, pelo barramento *Main Bus*.

6.1.1.1 Main Bus

A Main Bus consiste em um barramento principal interconectando todos os FPGAs Virtex 4 por meio de um barramento de 40 bits de largura, porém a interface Main Bus utiliza apenas 36 bits, sendo 4 bits de controle e 32 bits de dados, conforme mostrado na Figura 18 utilizando a sigla MB. Esse barramento pode ser usado para comunicação com interfaces USB, PCI ou PCI-e, dependendo do modelo da placa e de suas funcionalidades. Para a transferência de dados, cada palavra contém 32 bits representados em números hexadecimais e com o mesmo tamanho para realizar o endereçamento [DIN08a] [DIN08b]. O acesso a interface Main Bus pode ser realizada através de leitura ou escrita, respectivamente com as transações RD e WR, definidas nas Seções seguintes. Podem-se citar algumas características importantes da interface Main Bus [DIN08b]:

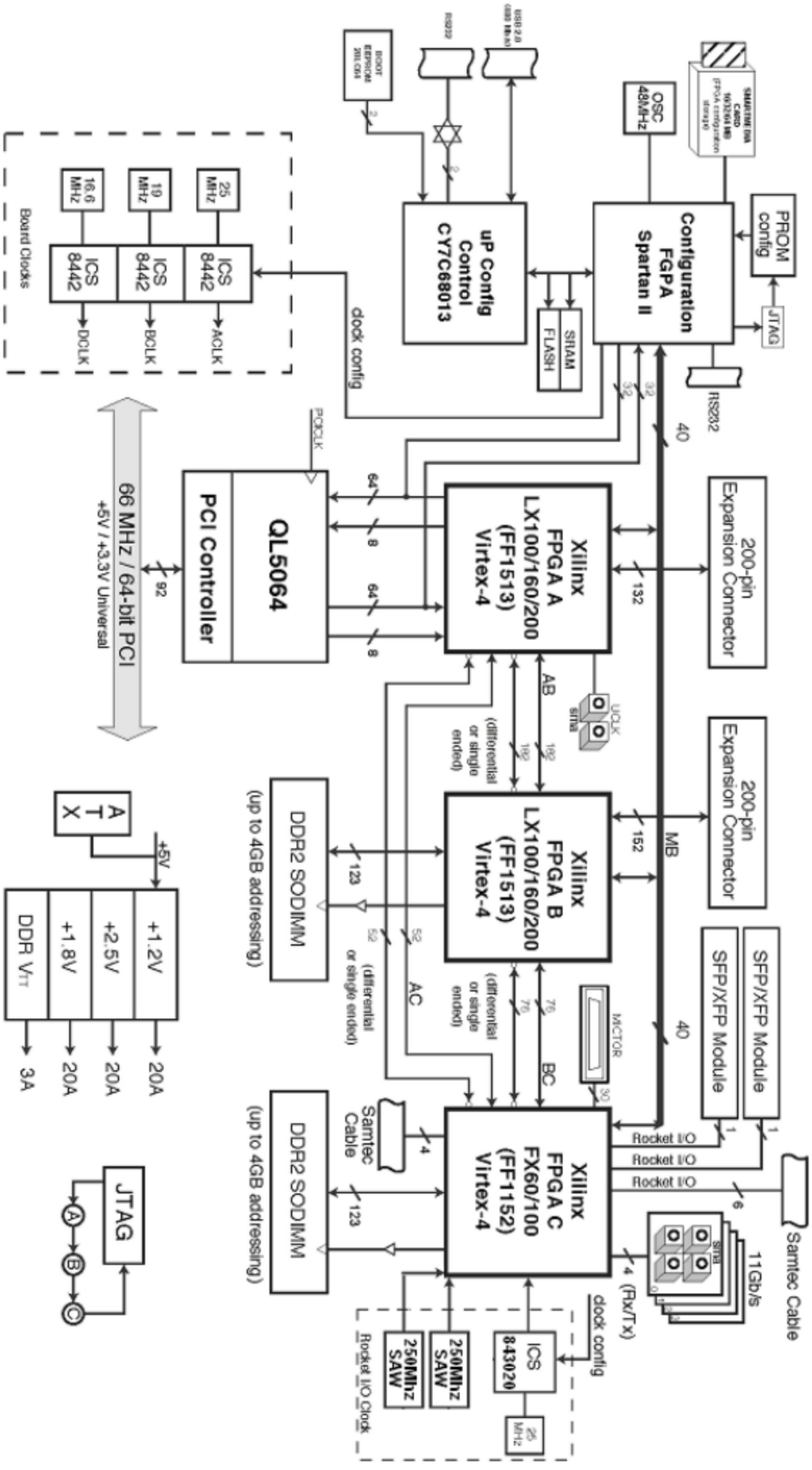


Figura 18 – Plataforma de Hardware modelo DN8000K10PCI, base para esse trabalho e fabricada pelo grupo DINI. Possibilita inserção de até 3 FPGAs, 2 soquetes de memória RAM (DDR2) com no máximo de 4GB e interfaces PCI, USB e RS232, são algumas características desta plataforma [DIN08a].

- Os acessos são sempre iniciados pelo FPGA Spartan II
- Suporte 1 circuito principal e 16 escravos
- Existem quatro sinais de controle: ALE, RD, WR e DONE
- Todas as transferências são síncronas conforme sinal MB_CLK, que é fixo em 48 Mhz.

6.1.1.1.1 RD

O RD é utilizado para realizar um acesso de leitura pelo barramento Main Bus, este processo consiste em três passos principais que são gerenciados pelo circuito principal, FPGA Spartan II. Primeiro, o sinal ALE é habilitado e o circuito escravo é selecionado conforme endereço, que requisita ao registrador os dados no barramento AD, depois o próximo ciclo de relógio é habilitado o sinal de leitura RD. Por último, o dispositivo principal espera habilitar o sinal DONE, e no mesmo ciclo de relógio coloca os dados no barramento AD para ser feita a leitura. A forma de onda da Figura 19 mostra todo esse processo [DIN08a].

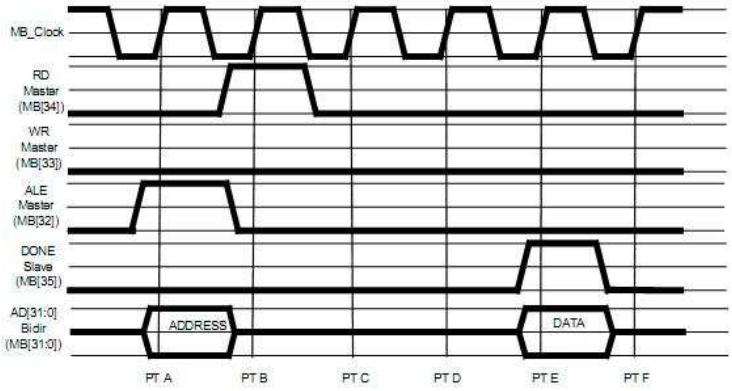


Figura 19 – Forma de onda da transação de leitura RD no barramento Main Bus com todos os sinais de controle utilizados [DIN08b].

6.1.1.1.2 WR

Como o RD, o WR pode efetuar acesso a interface Main Bus também gerenciado pelo dispositivo principal, FPGA Spartan II. No início, a seleção do dispositivo escravo é igual ao processo de leitura, mas após habilitar o sinal WR, no mesmo ciclo de relógio, é realizada a escrita dos dados no barramento AD. Finalmente, o dispositivo principal fica aguardando habilitar o sinal DONE para finalizar a transação e ficar pronto para uma próxima transferência, conforme a Figura 20 [DIN08a].

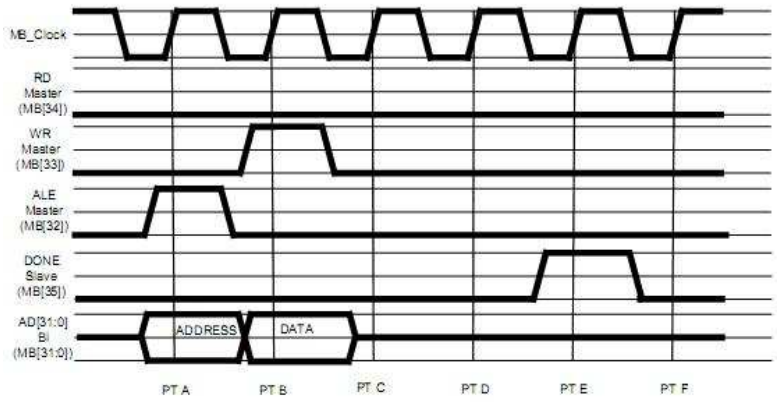


Figura 20 - Forma de onda da transação de leitura WR no barramento Main Bus com todos os sinais de controle utilizados [DIN08b].

6.1.1.2 Driver da Placa MDN8000K10PCI

O driver da placa é o *software* responsável por fazer a *interface* entre *hardware* e *software*, escrito em linguagem C, que é compilado e disponibilizado, pelo Dinigroup [DIN09], em conjunto com o *software* Aetest, descrito na Seção 6.1.3, em diversas plataformas de sistemas operacionais. Uma das plataformas suportadas é o Linux, que é a escolha desse trabalho, pois a aplicação por DM executa apenas neste sistema operacional. Esse *driver* possui suporte para o *kernel* 2.6 do Linux, porém houve problemas de compilação com o sistema operacional, na distribuição Fedora em suas duas últimas versões, que utilizam *kernel* acima do 2.6.25. Já com a versão 8, e *kernel* 2.6.23, não ocorreu o mesmo problema, lembrando que a arquitetura utilizada foi a i386 em todas as situações e conforme o fabricante este *software* tem suporte à arquitetura x86_64, não utilizada pois a máquina hospedeira tem processador com arquitetura de 32 bits. O fabricante também disponibiliza scripts para carregar o *driver*, mas estes tiveram que ser adaptados para funcionar. Quando o *driver* é carregado, cria 6 entradas de dispositivos com o nome dndev em /dev, entradas estas utilizadas no *software* Aetest para comunicação.

6.1.2 Projeto de Hardware

No grupo GAPH [GAP08] foi desenvolvido, no escopo de outro projeto de pesquisa, *hardware* que será utilizado aqui para realizar testes de leitura e escrita via interface PCI no barramento Main Bus, como ilustra a Figura 21. O módulo principal é o MB_Target, cuja função é comunicar-se com o dispositivo principal e gerenciar módulos escravos nas transações de leitura e escrita. Além deste módulo, o projeto contém um módulo para leitura de dados e um para escrita, que fazem acesso à memória pelos Handlers de cada transação. Por último, o módulo TOP faz manipulação dos dados. No presente projeto realiza-se uma transferência dos dados da memória de entrada para a memória de saída, não executando nenhum tipo de cálculo.

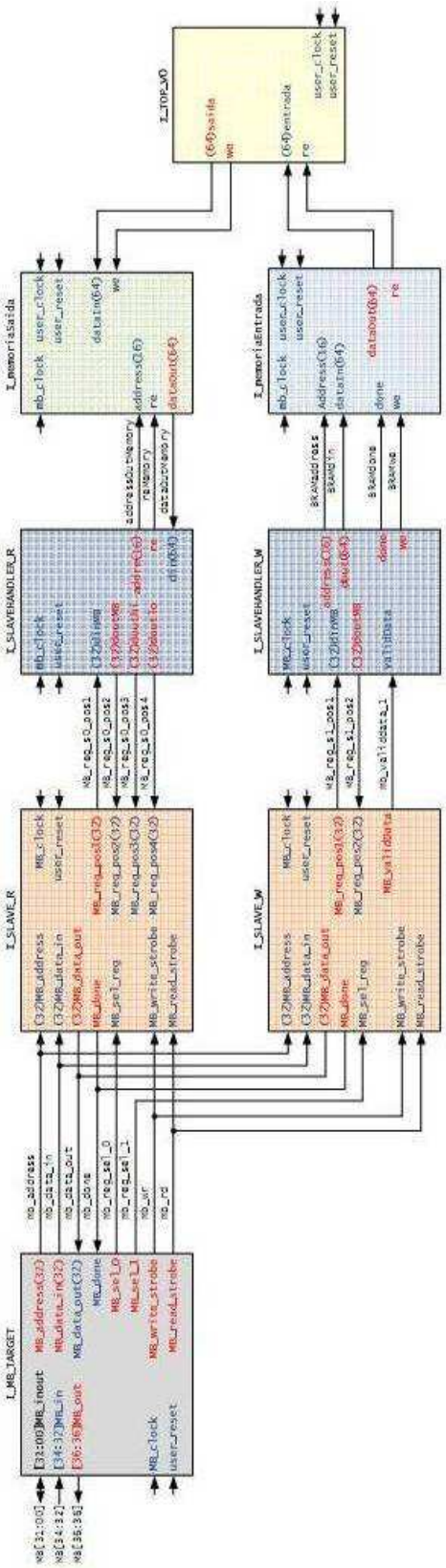


Figura 21 – Projeto de Hardware para efetuar transferências de dados pelo barramento Main Bus, utilizando o MB_Target e dois escravos, um para leitura e outro para a escrita de dados no FPGA. O armazenamento dos dados é realizado pelas *Block* RAMs [GAP08].

A transferência dos dados realizada por esse projeto é definida pelo envio ou recepção de 64 bits de dados por chamada e a memória (formada por *Block*RAMs do FPGA) total de armazenamento é de 81.600 palavras de 32 bits em hexadecimal. O fluxo de dados na entrada e saída desse projeto de hardware pelas chamadas de escrita e leitura no barramento Main Bus, é determinada pela técnica de ordenação FIFO (do inglês, *First In, First Out*).

Este *hardware* é a necessidade atual para efetuar os testes com a api nas transferências de dados entre a máquina hospedeira e a plataforma FPGA. Em trabalhos futuros esse *hardware* será aprimorado para efetuar os cálculos a serem acelerados na aplicação por DM.

6.1.3 Aetest

O fabricante da placa DN8000K10PCI, o Grupo Dini [DIN09], disponibiliza este *software* para executar diversos testes em suas plataformas FPGAs. Além disso, esse programa utilitário tem suporte a diversas plataformas de sistemas operacionais e diferentes funcionalidades como: teste na interface PCI, testes de memória SRAM, DDR, testes de alcance do barramento de memória, vários outros testes e verificações no *hardware*, entre outros.

O *software* Aetest contém duas funções essenciais para a necessidade deste trabalho, a mb_read e a mb_write, que possibilitam as transações de leitura e escrita no barramento *Main Bus*, definidas nas Seções 6.1.1.1.1 e 6.1.1.1.2. Estas funções estão implementadas no arquivo mdn8000k10pci.c, em alto nível de abstração, e faz chamadas às funções bar_write_dword() e bar_read_dword() localizadas no arquivo os_dep.c para depois comunicar-se diretamente com o *driver* da placa, estes arquivos fontes estão contidos no Aetest.

6.1.3.1 mb_write

Esta função realiza o acesso de escrita pela interface PCI pelo barramento Main Bus. Que passa como parâmetro duas informações, o endereçamento e os dados a serem escritos. Os dois parâmetros são do mesmo tipo de dado, *unsigned int* (padrão linguagem C), conforme protótipo da função na Figura 22.

```
void mb_write (unsigned int addr, unsigned int data)
```

Figura 22 – Função mb_write para escrita no barramento

Os tamanhos da palavra para os dados e endereços são de 32 bits, conforme Seção 6.1.1.1, representado pelas variáveis *addr* e *data* no protótipo da função. A palavra de endereçamento é definida na Figura 23, conforme especificação da interface *Main Bus*, já a palavra de dados é toda para o envio de dados e não contém nenhum tipo de controle.

Na Figura 23, o campo FPGA guarda o endereço do FPGA destino dentre os três possíveis na placa MDN8000K10PCI, o campo escravo seleciona o módulo escravo definido no projeto, podendo ter no máximo 16 possibilidades. Por fim, o campo endereços determina uma estrutura interna do módulo escravo que deverá armazenar os dados.

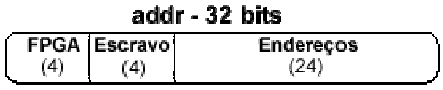


Figura 23 – Definição dos campos de controle e os tamanhos (bits) para o endereçamento na variável addr.

O trecho de código da Figura 24, em linguagem C, representa a função mb_write com seus comandos no arquivo mdn8000k10pci.c. A função possui basicamente quatro comandos que são chamadas a função *bar_write_dword*, e cada chamada possui uma função conforme a sequência da linha 3 à 6:

1. Habilita o acesso a interface PCI;
2. Habilita o acesso ao barramento Main Bus;
3. Define os endereços no barramento Main Bus;
4. Executa a escrita no barramento Main Bus.

```
1.      void mb_write(unsigned int addr, unsigned int data)
2.      {
3.              bar_write_dword(PCI_BAR,WRDATA,PCIENABLE);
4.              bar_write_dword(PCI_BAR,WRDATA,MBENABLE);
5.              bar_write_dword(PCI_BAR,MBADDR,addr);
6.              bar_write_dword(PCI_BAR,MBWRDATA,data);
7.      }
```

Figura 24 – Código em linguagem C da função MB_Write disponibilizado no software Aetest no arquivo referente à placa MDN8000K10PCI

Para acesso aos barramentos no driver em nível menor de abstração a função mb_write precisa passar pelo arquivo os_dep.c, conforme Figura 30, isso é feito pela função *bar_write_dword*, que possui o protótipo da Figura 25. Essa função possui três parâmetros com o mesmo tipo de dado *unsigned int*, e o tipo *dword* foi definido com esse mesmo tipo no arquivo *header*. Os dois primeiros parâmetros são para identificar e definir o barramento e o último para o envio dos dados.

```
int bar_write_dword(unsigned int barnum, unsigned int byte_offset, dword data)
```

Figura 25 - Função de baixo nível bar_write_dword para escrita no barramento

A mb_write é uma função simples e essencial na transferência de dados pelo barramento PCI e necessária para execução desse trabalho. As informações de endereços e dados são representadas em hexadecimal conforme especificação da própria Main Bus.

6.1.3.2 mb_read

A mb_read é uma função que realiza a leitura de dados pela interface PCI utilizando o barramento Main Bus para acesso ao *hardware*. Diferente da mb_write, só possui um parâmetro de entrada, o endereço para leitura dos dados. Outra diferença em relação ao mb_write, é que o mb_read possui retorno de valor *unsigned int*, dados a serem lidos. Conforme protótipo na Figura 26, o tipo de dado é o *unsigned int*, de 32 bits, representado em números hexadecimais.

```
unsigned int mb_read(unsigned int addr)
```

Figura 26 - Função mb_read para leitura no barramento

Figura 27 - Função de baixo nível bar_write_dword para escrita no barramento

Os campos de controle parra o endereçamento sobre a variável *addr* é o mesmo da função mb_write na Figura 23. O trecho de código que representa a função mb_read, está na Figura 28, que além do citado na função mb_write contém a inclusão da linha 7, com uma chamada *bar_read_dword* para a leitura de dados pelo barramento Main Bus.

```
1.  unsigned int mb_read(unsigned int addr)
2.  {
3.          unsigned int data;
4.          bar_write_dword(PCI_BAR,WRDATA,PCIENABLE);
5.          bar_write_dword(PCI_BAR,WRDATA,MBENABLE);
6.          bar_write_dword(PCI_BAR,MBADDR,addr);
7.          bar_read_dword(PCI_BAR,MBRDDATA,&data);
8.          return data;
9.  }
```

Figura 28 - Código em linguagem C da função MB_Read disponibilizado no software Aetest no arquivo referente à placa MDN8000K10PCI

A função bar_read_dword possui o protótipo da Figura 29, que diferente da *bar_write_dword*, contém um ponteiro no parâmetro referente aos dados a serem lidos.

```
int bar_read_dword(unsigned int barnum, unsigned int byte_offset, dword* data)
```

Figura 29 - Função de baixo nível bar_read_dword para leitura no barramento.

6.1.4 Módulo pmemd_clib

Na aplicação por DM, PMEMD pertencente ao pacote AMBER [AMB09], possui em sua maioria arquivos fontes baseados em linguagem Fortran e em específico o módulo pmemd_clib está escrito em linguagem C. Esta integração de linguagens existente nessa aplicação, permite o emprego de chamadas no código Fortran para C, facilitando a implementação em linguagem C padrão desse trabalho. A escolha da linguagem C para a implementação da API é devido ao *driver* da placa e a aplicação por DM utilizar esta linguagem, servindo como *interface* entre a máquina hospedeira e a plataforma de *hardware*, proposta da API.

A partir do módulo pmemd_clib serão acopladas todas as rotinas da API de comunicação entre o *hardware* e a aplicação por DM. Desta forma, é possível a integração com o *software* Aetest e consequentemente ao *driver* da plataforma de *hardware*, isso pode ser observado na Figura 30.

Na camada Aetest contém os arquivos necessários para a implementação da comunicação pela interface PCI, que fazem um intermediário entre a aplicação por DM e a parte de *software* de nível mais baixo, *driver* da plataforma de hardware, e utilizam os arquivos: mdn8000k10pci, pci e os_dep.

A forma como está organizado os arquivos fontes possibilita posteriores inserções de plataformas FPGAs compatíveis com o *driver* da placa DN8000K10PCI, pelo barramento PCI. Este *driver* possui suporte a diversas plataformas, fabricadas pelo grupo DINI, tendo a necessidade de apenas acoplar alguns parâmetros do modelo específico da plataforma de hardware no arquivo pci.

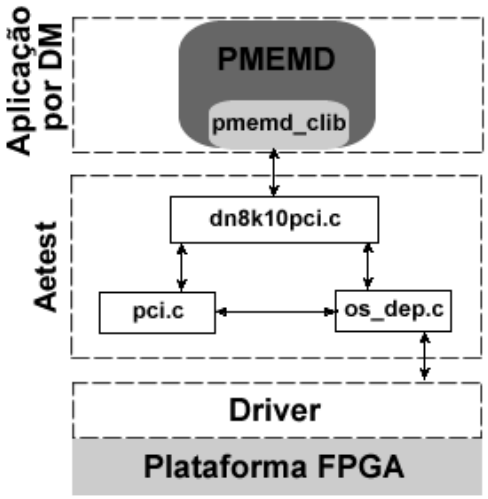


Figura 30 – Organização dos módulos envolvidos na API conforme nível de abstração e comunicação.

6.2 Compilação, Link-edição e Otimização

Após os testes preliminares com a plataforma FPGA (mdn8000k10pci) para leitura e escrita pela interface PCI, observou-se a quantidade de arquivos fontes e funcionalidades no *software* Aetest desnecessárias para o propósito do trabalho. Com isso, fez-se uma análise em todas as funcionalidades do *software* e seus fontes para otimizá-lo conforme a necessidade.

O pacote AMBER, versão 9, é disponibilizado para o sistema operacional Linux e o *software* Aetest oferecer suporte em diversos sistemas operacionais. Como o trabalho é voltado para a aplicação PMEMD, pertencente ao AMBER, não haveria necessidade do Aetest dar suporte a outros sistemas operacionais. Este foi um dos critérios encontrados para a otimização do código do Aetest. Portanto, todas as funcionalidades que não eram necessárias para a realização do trabalho

foram desabilitadas, para que o software Aetest se tornasse mais otimizado. Outros critérios foram utilizados para fazer a reorganização do código, como as verificações e testes em barramentos e memórias, operações feitas com memória, utilização de DMA, entre outros. Além do grande número de arquivos reduzidos, os que sobraram, também foram reduzidos em seus códigos internos, deixando apenas o essencial ao trabalho e a plataforma de sistema operacional adotada.

O *software* Aetest contém 52 arquivos sem contar os relacionados ao driver da placa. Este número foi reduzido para apenas 5 arquivos, porém houve a necessidade de executar diversas alterações no código original como:

- Conversão do código original em linguagem C++ para linguagem C padrão, visando o acoplamento no *driver* da placa e na aplicação por DM;
- Adaptação dos *Headers* para os novos arquivos fontes;
- Adaptação do *Makefile* para os novos arquivos e compreensão do mesmo para posterior acoplamento na aplicação por DM;
- Modificar o suporte de plataforma direcionando a aplicação por DM (Linux), retirando o suporte multiplataforma e definindo apenas monoplataforma.

A conversão de linguagens de programação do *software* Aetest foi necessária para acoplar o *driver* da placa MDN8000K10PCI com a aplicação por DM. Além disso, o pacote AMBER possui suporte as linguagens C e Fortran, facilitando a implementação de driver, que geralmente utilizam linguagem C padrão. Na compilação da aplicação de DM com o *software* Aetest, em linguagem C++, não foi possível pela quantidade de erros gerados na fase de linkedição dos objetos das linguagens contidas no PMEMD. Há literaturas que mostram a mistura das linguagens C++ e Fortran [NER09], e a linguagem C++ já possui a linguagem C padrão embutida, porém os parâmetros de compilação e linkedição da aplicação de DM e os compiladores utilizados foram fatores determinantes para impossibilitar a união das linguagens. Executaram-se diversos testes para esta união, porém não se obteve sucesso em nenhum deles. Com isso, a opção determinada foi a de migração do código da linguagem C++ para linguagem C.

Os arquivos intermediários da Figura 30, pertencentes ao Aetest, foram alterados para dar suporte as funcionalidades da API e a portabilidade. Houve a necessidade de adicionar no makefile e no arquivo de configuração da aplicação por DM, os devidos parâmetros de configuração e comandos para a compilação e linkedição dos novos arquivos fontes. Desta forma, conseguiu-se fazer a união do pacote AMBER com os arquivos da API e seus respectivos *headers*.

6.3 Novas Características Encontradas na Implementação

Observou-se na implementação da API, algumas características importantes ao trabalho

nesta fase. Com a mudança de arquitetura de *hardware* da máquina hospedeira no sistema operacional, de 64 para 32 bits, pelo fato da plataforma FPGA disponível estar em uma máquina com processador de 32 bits e a análise de perfil realizada na Seção 5.1 ter sido de 64 bits, ocorreram mudanças no código da função *short_ene* gerada pelo *software* PMEMD na fase de compilação.

A função *short_ene* utilizada em 64 bits era chamada de *Short_ene_novec*. Já em 32 bits a função chama-se s*hort_ene_vec*. Porém, além do nome, também foi alterado parte de seu código fonte. Assim, efetuou-se uma nova análise de perfil, mostrada na Figura 31. Se esta for comparada com a análise anterior (Figura 13), nota-se claramente as principais diferenças, na quantidade de laços e seleções. Na Figura 31, apresenta-se entre colchetes na função *short_ene_vec*, onde se encontra o PCH e quantidade de repetições em cada laço. As chamadas e a ordem das rotinas antecessoras a *short_ene* não foram alteradas. As características da simulação é igual a apresentada na Seção 5.1.

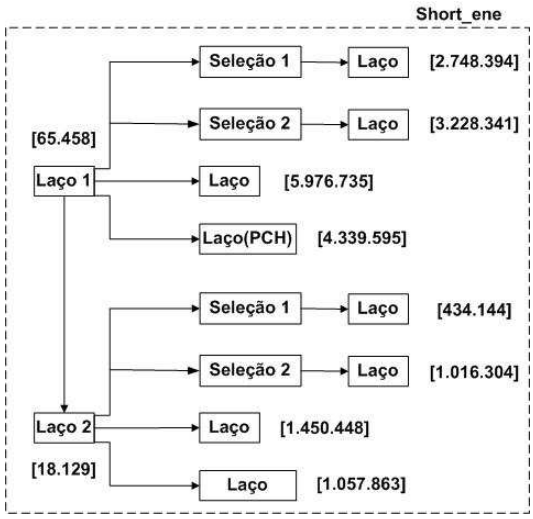


Figura 31 – Descrição da função *short_ene_vec* com todos os seus laços e suas respectivas quantidades de repetições.

Com o intuito de especificar com maiores detalhes a aplicação PMEMD, a função *short_ene_vec* e o laço PCH, fizeram-se necessária a alteração do código do PMEMD para demonstrar a quantidade de chamadas e o custo dessas partes do código no desempenho global da aplicação. Assim, a função *short_ene* e o laço PCH estavam como funções internas da sub-rotina *get_nb_energy* e foram alterados para sub-rotinas do módulo PME_DIRECT em duas simulações distintas, utilizando a ferramenta de traçado de perfil GPROF para adquirir essas informações.

Como já discutido e mostrado a literatura, anteriormente na Seção 5.1.1, a função short_ene_vec é a mais custosa em toda a simulação como pode ser observado na Figura 32 a sua representação no tempo global da simulação. Da mesma forma, o laço PCH também é o mais custoso dentro da função short_ene_vec e consequentemente da aplicação, Figura 33. Esses tempos

foram comprovados pela ferramenta GPROF com os mesmos parâmetros de simulação nas duas situações. Na Figura 32 apresenta apenas uma parte do relatório gerado pela ferramenta de traçado de perfil, no entanto, já pode ser comprovado os custos computacionais discutidos. A função short_ene_vec representa 75,47 % e o laço PCH, corresponde a 45,78% do tempo global da simulação com o software PMEMD, respectivamente mostrados nas Figura 32 e Figura 33. Na Figura 32 pode ser observado a relação entre a quantidade de chamadas efetuadas na função *short_ene_vec* com a do monitoramento do código, dividindo esse número pela quantidade de chamadas ao módulo PME_DIRECT, obtem-se uma quantidade similar ao da Figura 31.

% time	cumulative seconds	self seconds	self calls	total s/call	total s/call	name
75.47	195.67	195.67	14272400	0.00	0.00	pme_direct_modget_nb_energy_mp_short_ene_vec_
9.26	219.69	24.02	55	0.44	0.44	nb_pairlist_mod_mp_get_nb_list_
1.68	224.05	4.36	200	0.02	0.02	pme_recip_mod_mp_grad_sum_
1.51	227.96	3.91	200	0.02	0.02	pme_recip_mod_mp_fill_charge_grid_
1.16	230.97	3.01	200	0.02	0.02	pme_recip_mod_mp_scalar_sumrc_
0.84	233.15	2.18	1	2.18	258.62	runmd_mod_mp_runmd_
0.66	234.86	1.71	200	0.01	0.01	pme_force_mod_mp_nb_adjust_

Figura 32 – Perfil da Aplicação PMEMD, com o tempo cumulativo da função *short_ene_vec*.

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	self calls	total s/call	total s/call	name
45.78	142.37	142.37	26330108	0.00	0.00	pme_direct_modget_nb_energy_mp_laco_pch_
33.92	247.84	105.47	14272400	0.00	0.00	pme_direct_modget_nb_energy_mp_short_ene_vec_
7.64	271.60	23.76	55	0.43	0.43	nb_pairlist_mod_mp_get_nb_list_
1.41	275.98	4.38	200	0.02	0.02	pme_recip_mod_mp_grad_sum_
1.24	279.84	3.86	200	0.02	0.02	pme_recip_mod_mp_fill_charge_grid_
0.97	282.86	3.02	200	0.02	0.02	pme_recip_mod_mp_scalar_sumrc_
0.72	285.09	2.23	1	2.23	310.27	runmd_mod_mp_runmd_
0.56	286.82	1.73	200	0.01	0.01	pme_force_mod_mp_nb_adjust_

Figura 33 - Perfil da Aplicação PMEMD, com o tempo cumulativo do laço PCH.

Para se construir as primitivas da API foram necessárias investigações do código para a determinação dos dados a serem enviados e recebidos da plataforma FPGA. Os dados que serão enviados contará com as informações necessárias para a execução do PCH em *hardware*. A Tabela 4, mostra todos esses dados com seus respectivos tamanhos, tipos, tipo de conjunto de dados e módulos onde foram declaradas. As quantidades de variáveis em alguns dados, ef_tbl e eed_cub, podem variar dependendo do sistema molecular utilizado ou de diferenças na configuração da dinâmica, por serem alocados dinamicamente.

Tabela 4 – Características das variáveis para o envio à plataforma de hardware.

Nome da Variável	Tipo de Dado	Tipo de Conjunto de dados	Local de Inicialização (Módulos)	Quantidade de Variáveis
dens_efs	Double	Variável	short_ene_vec.i	1
CGI	Double	Variável	short_ene_vec.i	1
Img.charge	Double	Estrutura de Dados	Img.fpp	35.681
Nxt	Integer	Vetor	short_ene_vec.i	128
Img_j_vec	Integer	Vetor	short_ene_vec.i	128
Delr2_vec	Double	Vetor	short_ene_vec.i	128
Ef_tbl	Double	Vetor	ene_frc_splines.fpp (efs_tbl)	35.800
eed_stk	Double	Variável	pme_direct.fpp	1
eedvir_stk	Double	Variável	pme_direct.fpp	1
Dxdr	Double	Variável	pme_direct.fpp	1
Eedtbdns_stk	Double	Variável	pme_direct.fpp	1
Del	Double	Variável	pme_direct.fpp	1
Eed_cub	Double	Vetor	Pme_force	26.396
Del_vec	Double	Matriz	short_ene_vec.i	3 x 128
Img_frc	Double	Matriz	Pme_force	3 x 35.681

As variáveis que serão criadas ou atualizadas na plataforma de *hardware* e que necessitam retornar à aplicação por dinâmica molecular para dar continuidade a execução do *software*, são determinadas na Tabela 5. Para esses dados específicos, terá na API uma primitiva para a operação de envio desse conjunto de dados, exposto Tabela 4, e outra primitiva para a recepção dos dados referentes à Tabela 5.

Tabela 5 - Características das variáveis a serem recebidas da plataforma de hardware.

Nome da Variável	Tipo de Dado	Tipo de Conjunto de dados	Local de Inicialização (Módulos)	Quantidade de Variáveis
Sublst_head	Integer	Variável	Short_ene_vec.i	1
Eed_stk	Double	Variável	pme_direct.fpp	1
Eedvir_stk	Double	Variável	pme_direct.fpp	1
vxx	Double	Variável	pme_direct.fpp	1
vxy	Double	Variável	pme_direct.fpp	1
vxz	Double	Variável	pme_direct.fpp	1
vyy	Double	Variável	pme_direct.fpp	1
vyz	Double	Variável	pme_direct.fpp	1
vzz	Double	Variável	pme_direct.fpp	1
dumx	Double	Variável	Short_ene_vec.i	1
dumy	Double	Variável	Short_ene_vec.i	1
dumz	Double	Variável	Short_ene_vec.i	1
Img_frc	Double	Matriz	Short_ene_vec.i	3x35.681

7 PROPOSTA DE ARQUITETURA DE SOFTWARE

A arquitetura de *software* proposta nesse trabalho é de extrema importância a definição da forma e da estrutura das chamadas à API de comunicação entre *hardware* e *software*. Ressalta-se a importância da organização desta arquitetura de *software*, para abordar a comunicação entre *hardware* e *software*, para que seja possível a atuação em diferentes níveis de abstração.

Para conter um nível de abstração coerente e com intuito de facilitar a utilização da API, determinou-se o emprego de 3 níveis de abstração. O primeiro nível trata-se da camada superior do *software*, onde o programador efetuará as chamadas à API. No segundo, foi determinada a *interface* de comunicação utilizada pela plataforma de *hardware* como PCI, *Ethernet*, USB, entre outras. O último nível possui as plataformas de *hardware* a serem utilizadas dependendo da *interface* definida anteriormente. Com essa divisão pode-se adaptar funcionalidades, plataformas de *hardware* e execução em sistemas operacionais distintos, conforme Figura 34. Assim, a facilidade em agregar funções conforme a necessidade de projeto torna-se uma tarefa mais simples, proporcionando portabilidade a API.

Os módulos adaptados ao *software* PMEMD, foram organizados de forma a apresentar uma estrutura conforme o nível de abstração inicialmente adotado como guia. Pode-se verificar esta estrutura e o nível de abstração, fazendo uma comparação entre a Figura 30 e a Figura 34, com foco na portabilidade da API. Assim, o primeiro nível relaciona-se ao nível do módulo pmemd_clib acoplado ao software PMEMD, o nível intermediário relacionado ao Aetest e o terceiro nível, mais baixo, relaciona-se ao *driver* da plataforma de hardware.

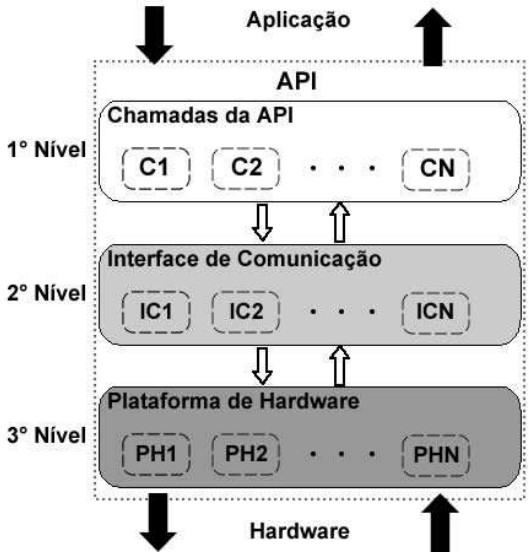


Figura 34 – Níveis de Abstração da API.

Dependendo da plataforma FPGA ou interface de comunicação a ser empregada no projeto, terá de ser modificado apenas o nível, em que o novo projeto se diferencia, tornando-se pequena complexidade desta integração com a aplicação de simulação por DM. A API será utilizada na aplicação PMEMD e com isso a portabilidade dela para outro *software* que não pertence ao pacote AMBER poderá ser realizada se tiver suporte a linguagem C, que é uma linguagem muito utilizada em aplicações científicas e que permite fácil mixagem de linguagem.

As rotinas para o envio e recepção com uma variável, um vetor ou uma matriz, foram construídas para facilitar o uso da API no processo de crescimento do projeto. Assim, o transporte de pequenos códigos da aplicação por DM para hardware, e/ou na fase de testes de comunicação entre a máquina hospedeira e a plataforma FPGA. Já as primitivas de manipulação de diversas variáveis, tornam-se específicas para a finalidade desse trabalho, cujo intuito é enviar um conjunto de dados necessários, para a execução do laço mais custoso computacionalmente no FPGA. A recepção dos dados atualizados é necessária para continuidade da aplicação por DM. A comunicação software/hardware, nestas rotinas, inicializa no carregamento do módulo do driver da plataforma de hardware, selecionando o modelo da plataforma a ser utilizada através de requisição ao driver. Após carregar o driver da plataforma podem ser executadas as rotinas que fazem a escrita ou leitura no barramento Main Bus conforme o tipo de primitiva. Inicialmente, definiram-se algumas primitivas que serão padronizadas na API conforme apresentadas na Tabela 6. Foi disponibilizado suporte a uma plataforma FPGA por meio de uma interface PCI de comunicação que melhor se adapta à necessidade e à disponibilidade atual. Poderão ser usadas outras plataformas com suporte a outras interfaces apenas com adaptação dos módulos adequados.

As primitivas que determinam todos os dados necessários à implementação do laço alvo na função short_ene são API_DATA e API_REC_DATA. Estas determinam os dados a serem transferidos ao FPGA. Estas rotinas têm seus dados diretamente relacionados ao código da função short_ene analisado e ao PCH escolhido. Abaixo se mostra o protótipo dessas duas rotinas, onde se encontram todas as variáveis com seus nomes originais na aplicação por DM precedido do tipo de conjunto de dados, como variável, vetor e matriz. Nesses protótipos pode-se observar o uso de ponteiros em todas as variáveis, independente do seu tamanho ou tipo. Isso se dá pela mistura entre Fortran e C, com dados tratados via ponteiros no PMEMD, o que facilita sua manipulação.

As rotinas apresentadas nesta Seção demonstram a padronização adotada na arquitetura de *software* criada e a portabilidade desta API, com o intuito de facilitar a sua utilização e disseminação. Na implementação desta arquitetura de *software*, criou-se um padrão para a codificação dos dados inteiros e de ponto flutuante, obedecendo aos critérios impostos no projeto de *hardware* descrito na Seção 6.1.2.

Tabela 6 – Descrição das primitivas da arquitetura de software.

ROTINA	**DESCRIÇÃO**
API_DATA (controle,dado*1*, ..., dado*15*)	Esta rotina permite o envio de um conjunto de dados necessários para execução do primeiro laço mais custoso da sub-rotina *short_ene*, tornando-se específica para o problema proposto. Contém nesta primitiva dados de vários tipos e conjuntos como variável simples, vetor e matriz, geralmente em ponto flutuante. Apenas o primeiro campo é utilizado para controle de execução desta primitiva, pode-se ter uma variável ou apenas um valor para este campo e apenas se este valor for igual a 1 será iniciada esta rotina. Este campo será definido em todas primitivas da mesma forma e na mesma posição.
API_DATA_I (controle, dado, parâmetro,tamanho)	Esta rotina permite o envio de dados do tipo inteiro para a plataforma FPGA. O primeiro campo foi determinado na primeira primitiva. O segundo campo determina os dados que se deseja enviar a plataforma de hardware. O terceiro campo pode-se definir o tipo de conjuntos de dados s serem enviados como variável simples, vetor ou matriz colocando os valores 0, 1 ou 2 respectivamente, tornando-se flexível, mas respeitando o tipo de dado. O quarto parâmetro define o tamanho do vetor, e caso seja matriz, será colocado o comprimento das linhas, pois a quantidade de linhas encontradas nas matrizes no PCH são as mesmas (três), definindo uma restrição. Porém, se houver necessidade pode-se parametrizar toda a matriz facilmente. Coloca-se 0 se a variável for comum. Os dados são representados com o tipo inteiro padrão, de tamanho 4 bytes definidos em linguagem C por *int*, dependendo da arquitetura do processador o tamanho pode variar.
API_DATA_D(controle,dado, parâmetro,tamanho)	Esta rotina permite o envio de dados com o tipo ponto flutuante de dupla precisão para a plataforma FPGA. Os parâmetros referenciados por esta primitiva seguem a mesma definição e restrições encontradas em API_DATA_I. Os dados são representados pelo tipo ponto flutuante preferencialmente com tamanho de 8 bytes definida em linguagem C padrão pelo tipo double. Inicialmente não será utilizada toda a precisão do ponto flutuante, será definido aproximadamente em 32 bits podendo chegar até 39 bits, conforme [GU08] apresenta que este range é suficiente para a necessidade da aplicação por DM, isto é comum a todas primitivas.
API_REC_DATA(controle, dado*1*, ..., dado*13*)	Depois de enviado os dados pela função API_DATA(), é necessário esperar o FPGA efetuar os cálculos e retornar apenas as variáveis necessárias para dar prosseguimento a execução da simulação no software PMEMD. A espera e recepção dos dados (variáveis) na máquina hospedeira pela aplicação por DM é realizada pela presente função. Contém nesta primitiva dados de vários tipos e conjuntos como variável simples, vetor e matriz, geralmente em ponto flutuante.
API_REC_I(controle, dado, param, tamanho)	Diferente da rotina API_REC_DATA(), esta primitiva recebe apenas uma variável, vetor ou matriz por vez e do tipo inteiro (int em linguagem c). O segundo campo da função determina a variável que receberá os dados da plataforma de hardware respeitando o tipo de dado. Os parâmetros referenciados por esta chamada seguem a mesma definição e restrições encontradas em API_DATA_I.
API_REC_D(controle, dado, param, tamanho)	Esta rotina recebe apenas uma variável ou um vetor por vez e do tipo ponto flutuante de dupla precisão, double em linguagem c, conforme definição em API_DATA_D. Os parâmetros referenciados por esta chamada seguem a mesma definição e restrições encontradas em API_REC_I, com diferença apenas no tipo de dado a ser recebido.

```
void API_DATA(int * controle, double * var_densefs, double *
          var_cgi, double * var_eedstk, double *
          var_eedvirstk, double * var_dxdr, double *
          var_eedtbdnsstk, double * var_del, double *
          str_imgcharge, int * vet_nxt, int * vet_imgjvec,
          double * vet_delr2vec, double * vet_eftbl, double *
          vet_eedcub, double * mat_delvec, double *
          mat_imgfrc)

void API_REC_DATA(int * controle, int * var_sublsthead, double
          * var_eedstk, double * var_eedvirstk, double *
          var_vxx, double * var_vxy, double * var_vxz, double
          * var_vyy, double * var_vyz, double * var_vzz,
          double * var_dumx, double * var_dumy, double *
          var_dumz, double * mat_imgfrc)
```

7.1 Resultados

Na validação da API em uma simulação por dinâmica molecular, trabalhou-se em conjunto com o software PMEMD. Desta forma, é possível o envio de dados em tempo de simulação, da aplicação por DM, para a plataforma de hardware e o retorno de dados para a aplicação por DM com a API fazendo a interface entre as plataformas, máquina hospedeira (PMEMD) e Plataforma de Hardware (FPGA).

Para demonstrar esta validação foram realizados dois experimentos preliminares com parâmetros iguais da simulação por DM, do projeto de hardware no FPGA e modos de transferência da API. Os parâmetros do sistema molecular e de entrada para a simulação por DM, descritos na Capítulo 2, são iguais aos utilizados nos estudos de caso, com a diferença de conter apenas um *time step* na simulação. Esta quantidade foi definida por já surgir efeitos de comparação nos dois experimentos.

A transferência de dados, seja receber ou enviar, é composta por uma chamada ao barramento com uma palavra de 32 bits em hexadecimal. Determinou-se que na API as variáveis do tipo inteiro utilizariam apenas uma palavra e as de ponto flutuante duas palavras. Assim, a cada transferência de dados teria no mínimo duas palavras, independente do tipo de dado a ser transferido para ser padronizado em todas as situações. A definição do início e término dos dados fica a cargo da codificação e decodificação de todos os dados para corresponder ao número hexadecimal empregado pelo barramento, realizando um protocolo próprio para identificação dos dados. Isso para comprovar a veracidade das informações transferidas entre as plataformas, obedecer à restrição do barramento e possibilitar a validação da API. Os tempos de execução demonstrados nos dois experimentos foram retirados do JAC benchmark do próprio software

PMEMD e corresponde aos tempos de execução de não CPU ou de entrada e saída.

7.1.1 Primeiro Experimento

Em um primeiro momento realizou-se uma implementação da API com o objetivo de alcançar o princípio básico da API de empregar a comunicação da máquina hospedeira e a plataforma de hardware. Assim, não houve qualquer preocupação em relação ao desempenho da comunicação entre as plataformas neste primeiro momento.

Para realizar a comunicação foi preciso inserir as primitivas da API no código do software PMEMD, em linguagem Fortran, com suas devidas características. Assim, as operações de leitura e escrita foram acionadas pelo primeiro nível da API, passando pelo nível intermediário com as definições básicas de sistema operacional e definições dos endereçamentos no barramento, e no terceiro nível, realizou-se a inicialização do *driver* e o mapeamento dos barramentos, conforme Figura 35.

Nesta primeira etapa definiu-se que para cada envio ou recepção de duas palavras de 32 bits, os níveis dois e três seriam inicializados e desabilitados. Então, a cada duas palavras tem-se uma interrupção ao *driver* da placa MDN8000K10PCI para o envio ou recepção destas palavras. A Figura 35 mostra com detalhes a chamada de envio de dados do tipo inteiro. Nesta interrupção troca-se o modo de acesso de usuário para *kernel* para permitir o acesso ao dispositivo de hardware.

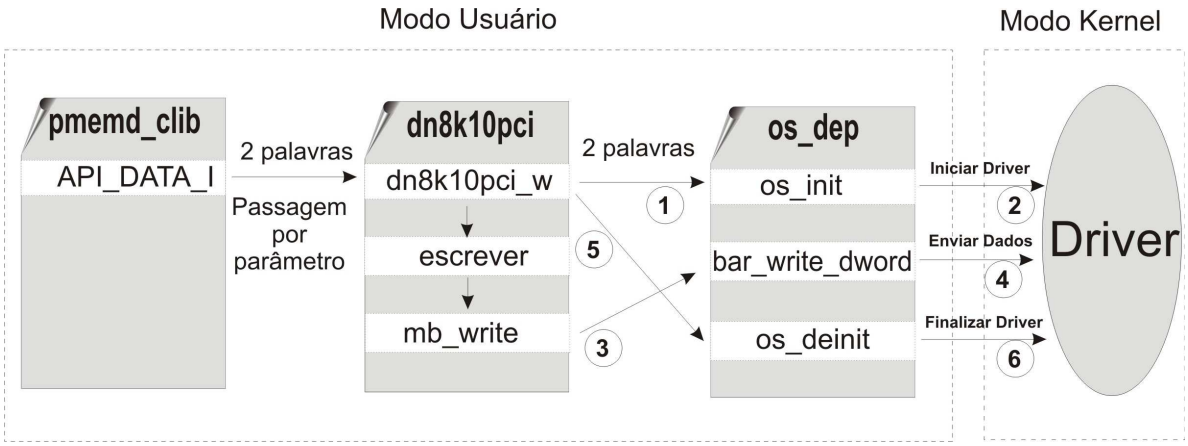


Figura 35 – Chamada da API de envio de dados do tipo inteiro, com o envio de duas palavras de dados por interrupção.

Nos experimentos demarcaram-se diferentes tamanhos e tipos de variáveis, e também os tipos de transferência realizados. Primeiro foram introduzidos os dois tipos de variáveis, inteiro e ponto flutuante, utilizadas no PCH do software PMEMD. Os tamanhos são com uma variável e três dimensões de vetores (8, 32, 128) e matrizes (3x8, 3x32, 3x128). Todas as matrizes do PCH possuem 3 linhas e por esse motivo empregou-se esse valor específico em dados referentes a

matrizes. Para facilitar a visualização, foram atribuídos os seguintes tamanhos 1, 8, 24, 32, 96, 128 e 384, sendo que os vetores correspondem aos números 8, 32 e 128, e as matrizes aos números 24, 96, 384. O valor zero na Figura 36 corresponde à execução da simulação no software PMEMD sem chamadas à API. Foram realizados três tipos de transferências, envio (SEND), recepção (REC) e envio e recepção (SEND_REC) para cada valor de quantidade de variáveis, conforme Figura 36. Pelo gráfico pode-se observar a semelhança nos tempos das operações entre SEND_I e REC_I, e entre as operações SEND_D, REC_D e SEND_REC_I. E em todos os casos, com uma quantidade maior que um, os tempos de transferência são inaceitáveis no aproveitamento do barramento.

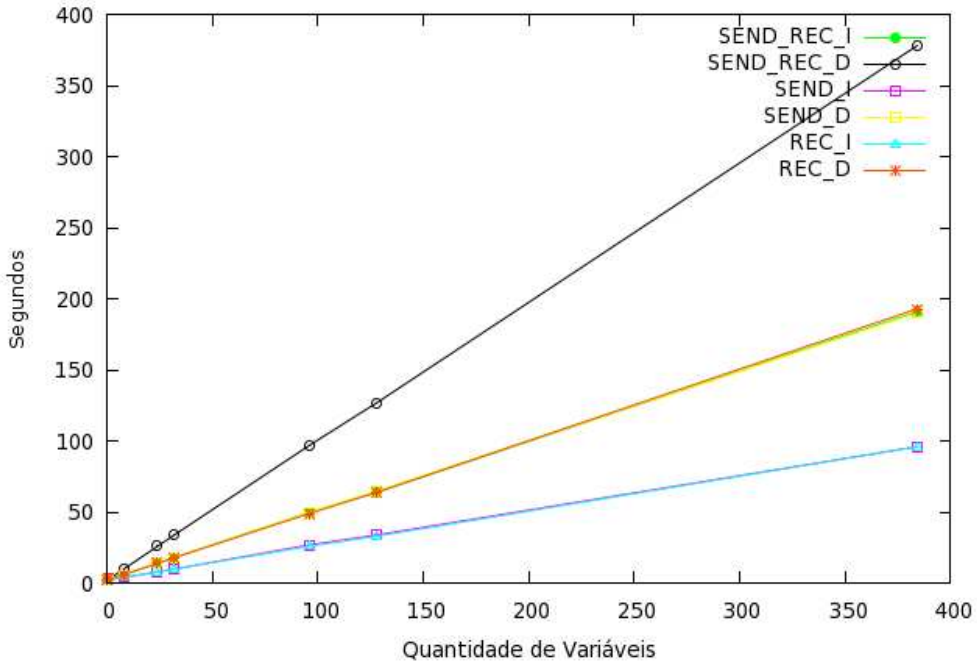


Figura 36 – Primeiro experimento com a API com diferentes tipos de transferência de dados e quantidades de variáveis relacionadas ao tempo de execução em cada definição. O último caractere em cada tipo de transferência corresponde ao tipo de dados enviado, I para inteiro e D para ponto flutuante, precisão simples ou dupla.

Com esse experimento comprovou-se a viabilidade de transferências entre as plataformas sem corrupção dos dados na API. Além disso, tornou-se clara a necessidade de ajustes na API, pois o tempo na realização das transferências é inaceitável, em vista do uso de um barramento PCI.

7.1.2 Segundo Experimento

Como no primeiro experimento observou-se um alto tempo de execução das operações de transferência de dados da API, este segundo experimento visou melhorar o desempenho da transferência em termos de tempo. Os resultados são apresentados na Figura 37 e na Figura 38.

Verificou-se que não havia necessidade de que cada transferência de duas palavras tivesse

uma interrupção que habilitava e desabilitava o *driver* da plataforma de hardware. Isso estava causando um grande aumento de tempo nas transferências realizadas com um maior número de variáveis e conseqüentemente um maior número de interrupções na plataforma de hardware.

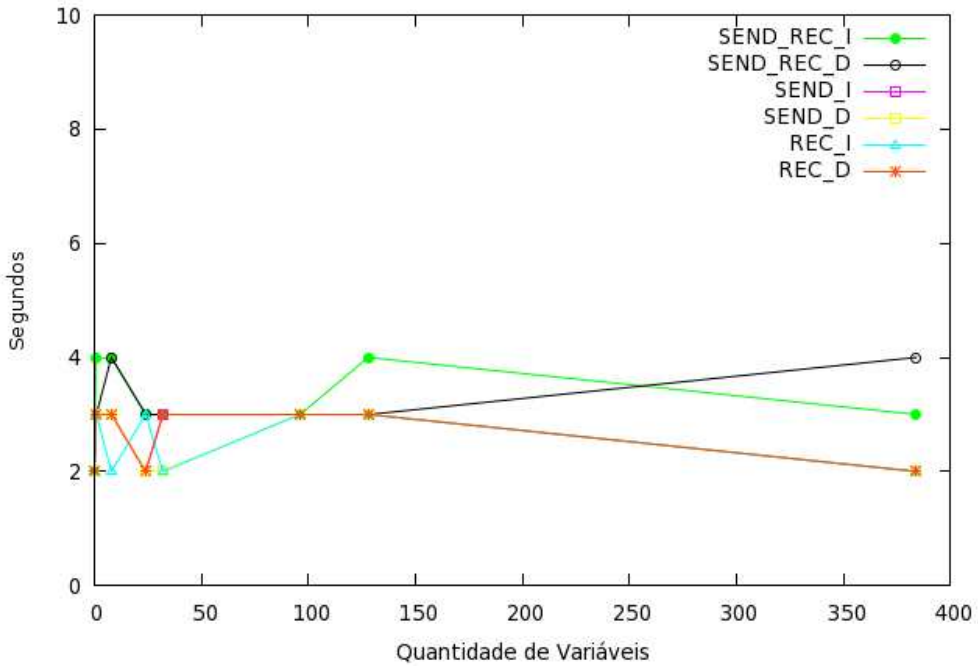


Figura 37 – Segundo experimento com escala reduzida para facilitar a visualização das variações nos tempos de execução.

Para garantir um tempo de transferência aceitável, a interrupção foi estabelecida para cada chamada feita à API. Com isso, independente da quantidade de variáveis a serem transferidas entre as plataformas o tempo gasto com a interrupção é o mesmo em todos os casos, utilizando alocação dinâmica de memória e passando apenas o ponteiro para quando for necessário o acesso aos dados a serem transferidos.

Na validação desta melhoria realizada pode-se observar na Figura 37, em que ocorre uma pequena variação, no tempo de execução, em cada operação. Isso ocorre, pois a própria simulação que possui quantidades diferentes no tempo de processamento influenciando no resultado apresentado. Os valores demonstrados na Figura 37 são adquiridos pelo JAC *benchmark* do próprio software de simulação por dinâmica molecular, PMEMD.

Reduziu-se a escala do tempo de execução na Figura 37 para que sejam visualizadas as variações ocorridas em cada operação, assim é possível verificar que em alguns casos as variações ocorrem em ordem decrescente da quantidade de variáveis. Desta forma, pode-se afirmar que a execução da API não está influenciando no tempo de execução da simulação e quem está determinando essas diferenças é a própria simulação. Vale lembrar que o tempo de execução com a quantidade de variáveis em zero, quando não há nenhuma chamada da API no código da aplicação.

Como cada operação em cada quantidade de variáveis é uma simulação distinta, foi realizada esta simulação sem a API, para efeito de comparação, nesse caso o tempo total da simulação ficou em 2 (dois) segundos, Figura 38. No entanto, existem vários momentos que mesmo com chamadas de transferências com diversas variáveis o valor permanece igual ao da simulação sem a API, mostrando que a API não está influenciando no tempo total de Non-CPU.

Na Figura 38 modificou-se a escala do gráfico para observar a simulação sem API, em 2 (dois), e as operações com até 32 variáveis no momento inicial do gráfico. Com isso, algumas chamadas a API,, com diversas quantidades de variáveis, ficam com o mesmo tempo de custo da simulação do momento em que não há nenhuma chamada a API, provando a sua validação.

Neste experimento é possível observar a validação da API com uma grande variedade de parâmetros e tempos de execução. E não houve nenhuma influência no tempo de simulação da molécula no software PMEMD. Os dados envolvidos foram testados no envio e no seu retorno para o software na máquina hospedeira sem nenhum problema, utilizando-se a plataforma de hardware, descrita na Seção 7.1.1, populado com um FPGA XCV4FX100 para validar a API.

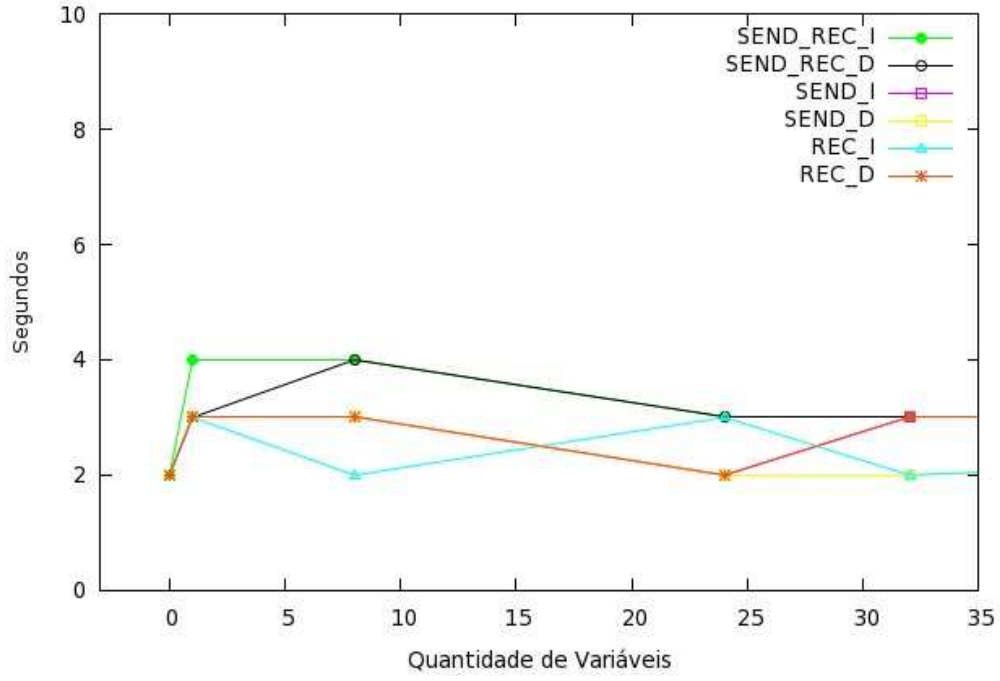


Figura 38 - Segundo experimento com escala reduzida e a parte inicial da quantidade de variáveis.

8 CONSIDERAÇÕES FINAIS

8.1 Conclusões

Este trabalho apresentou o desenvolvimento de uma arquitetura de *software* para comunicação de uma máquina hospedeira com uma plataforma de hardware baseada em FPGAs, para dar suporte ao modelo de organização de estrutura de alto desempenho com *hardware* dedicado.

Para tanto, nas seções 6.2 e 6.3, são apresentadas investigações de perfil da aplicação e de desempenho das aplicações por DM, por meio de experimentos realizados a fim de compreender o ambiente ao qual a proposta de arquitetura de software será empregada.

Na Seção 5.1, encontrou-se o PCH definido na Figura 1 com a ferramenta de *profiling* gprof pelo traçado de perfil e do gráfico de chamadas adquiridos na simulação por DM com o *software* PMEMD. Na Seção 5.2 efetuaram-se experimentos em diversos ambientes com os *softwares* PMEMD e SANDER comparando-os e escolhendo o PMEMD como *software* base pelos resultados dos experimentos e por dados adquiridos de outras literaturas como [EUR07][HEI05], demonstrando o seu desempenho superior ao SANDER. Decidiu-se na Seção 5.3 pela não utilização da biblioteca MPI pela alta quantidade de recursos exigidos por esta, bem como pelas restrições impostas a sua implementação em plataforma FPGA.

A arquitetura de *software* criada e testada, com os recursos disponíveis atualmente, será disponibilizada ao grupo GAPH para dar continuidade ao objetivo do projeto geral ou outros pertinentes. A API foi validada no Capítulo 7 com a demonstração de dois experimentos realizados por meio de diversos parâmetros diferentes de comparação.

Devido à diversidade de ferramentas, métodos, técnicas, recursos de *hardware*, cálculos matemáticos, entre outros, pode-se averiguar que a abrangência de áreas agregadas dificultou o entendimento e a visão do objetivo a ser alcançado. Finalmente, o trabalho e os resultados apresentados demonstram a pertinência da área escolhida como tema de estudo.

8.2 Trabalhos Futuros

Como a diversidade da área é de maneira vasta, podem-se determinar várias propostas a futuros trabalhos. A implementação da API em diferentes aplicações por dinâmica molecular que utilizam o método de replicação de dados, e acompanhado de um estudo maior em outros métodos de paralelização. A inclusão da API em outros projetos ou apenas com diferentes plataformas de

hardware baseadas em FPGAs utilizando recursos de HPRC.

A plataforma FPGA disponível para a realização desse trabalho possui uma interface PCI que apresenta seu melhor desempenho nesta plataforma. Porém, com a disponibilidade de plataformas mais recentes, onde se encontram interfaces como PCI-e, por exemplo, que tem um desempenho melhor do que a interface PCI, poder-se-ia adaptar a API para uma plataforma deste tipo ou de outros conforme a necessidade.

A arquitetura de *software* apresentada nesse trabalho pode ser adaptada para projetos de outras áreas, diferente de simulação por dinâmica molecular, pois apresenta transferência de dados com tipos de variáveis padrões. Porém, tem-se que respeitar a forma de passagem de parâmetros encontrada nesta API onde foi determinada pela mixagem da linguagem Fortran com a linguagem C determinada pela especificidade imposta do projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

[ADC06] ADCOCK, S. A.; MCCAMMON, J. A. "Molecular Dynamics: Survey of Methods for Simulating the Activity of Proteins". Journal Chemical Reviews, 106(5), 2006, pp. 1589-1615.

[AGA06] AGARWAL, P. K.; ALAM, S. R.; GEIST A.; VETTER, J. S. "Performance Characterization of Molecular Dynamics Techniques for Biomolecular Simulations". In: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06), 2006, pp. 59-68.

[AGA07a] AGARWAL, P. K.; ALAM, S. R.; SMITH, M. C.; VETTER, J. S. "Throughput Improvement of Molecular Dynamics Simulations Using Reconfigurable Computing". In: Scalable Computing: Practice and Experience (SCPE'07), 8(4), 2007, pp. 395-410.

[AGA07b] AGARWAL, P. K.; ALAM, S. R.; SMITH, M. C.; VETTER, J. S.; CALIGA, D. "Using FPGA Devices to Accelerate Biomolecular Simulations". IEEE Computer, 40(3), 2007, pp. 66-73.

[AHM00] AHMED, E.; ROSE, J. "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density". In: Eighth International Symposium on Field Programmable Gate Arrays, 2000, pp. 3-12.

[AMB08a] AMBER – Assisted Model Building with Energy Refinement. "AMBER 9 Users' Manual". Capturado em: http://ambermd.org/doc9/, Outubro 2008.

[AMB08b] AMBER – Assisted Model Building with Energy Refinement. "PMEMD 3.00 (Particle Mesh Ewald Molecular Dynamics) Release Notes". Capturado em: http://archive.ambermd.org/200404/att-0276/pmemd3.00.ReleaseNote, Outubro 2008.

[AMB09] AMBER – Assisted Model Building with Energy Refinement. "Amber Home Page". Capturado em: http://ambermd.org/, Fevereiro 2009.

[BAT07] BATSON, B.; BOWERS, K. J.; CHAO, J. C.; DENEROFF, M. M.; DROR, R. O.; EASTWOOD, M. P.; GROSSMAN, J. P.; GAGLIARDO, J.; HO, C. R.; IERARDI,

D. J.; KLEPEIS, J. L.; KOLOSSVÁRY, I.; KUSKIN, J. S.; LARSON, R. H.; LAYMAN, T.; MCLEAVEY, C.; MORAES, M. A.; MUELLER, R.; PRIEST, E. C.; SALMON, J. K.; SHAN, Y.; SHAW, D. E.; SPENGLER, J.; THEOBALD, M.; TOWLES, B.; YOUNG, C.; WANG, S. C. "Anton, a Special-Purpose Machine for Molecular Dynamics Simulation". In: The 34th Annual International Symposium on Computer Architecture (ISCA'07), 2007, pp.1-12.

[BEA06] BEAUCHAMP, M.; HAUCK, S.; UNDERWOOD, K.; HEMMERT, K. "Architectural modifications to improve floating-point unit efficiency in FPGAs". In: Proc. Field Prog. Logic and Applications, 2006, pp. 515–520.

[BEN02] BENINI, L.; DE MICHELI, G. "Networks on Chips: A New SoC Paradigm". IEEE Computer, 35(1), 2002, pp. 70-78.

[BER95] BERENDSEN, H.; VAN DER SPOEL, D.; VAN DRUNEN, R. "GROMACS: a message-passing parallel molecular dynamics implementation". Comput. Phys. Commun., 91, 1995, pp. 43–56.

[BET98] BETZ, V.; ROSE, J. "How Much Logic Should Go in an FPGA Logic Block?". In: IEEE Design and Test Magazine, 1998, pp. 10-15.

[BRO83] BROOKS, B. R.; BRUCCOLERI, R. E.; OLAFSON, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations". Journal Comput. Chem., 4, 1983, pp. 187–217.

[BUR94] BURNS, G.; DAOUD, R.; VAIGL, J. "LAM: An Open Cluster Environment for MPI". In: Proceedings of Supercomputing Symposium, 1994, pp. 379–386.

[BYS01] BYSTROFF, C.; GARDE, S. "Helix propensities of short peptides: Molecular dynamics versus Bioinformatics". Proteins: Structure, Function and Bioinformatics Journal, 50, 2001, pp. 552-562.

[CAS05] CASE, D. A.; CHEATHAM, T. E.; DARDEN, T.; GOHLKE, H.; LUO, R.; JR., K. M. M.; ONUFRIEV, A.; SIMMERLING, C.; WANG, B.; WOODS, R. J. "The Amber Biomolecular Simulation Programs". Journal Comput. Chem., 26(16), 2005, pp. 1668–1688.

[CON07] CONTI, A.; DISABELLO, D.; GU, Y.; HERBORDT, M. C.; MODEL, J.; SUKHWANI, B.; VANCOURT, T. “Achieving High Performance with FPGA-Based Computing”. In: International Symposium on Software Reliability Engineering (ISSRE’07), 40(3), 2007, pp. 50-57.

[COS06] COSTA, César da. “Projetando Controladores Digitais com FPGA”. Novatec Editora, 2006, pp. 27-38.

[DAR93] DARDEN, T. A.; YORK, D.; PEDERSEN, L. “Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems”. The Journal of Chemical Physics, 98(12), 1993, pp. 10089-10092.

[DAR99] DARDEN, T. A.; SAGUI, C. “Molecular Dynamics Simulations of Biomolecules: Long-Range Electrostatic Effects”. Annu. Rev. Biophys. Biomol. Struct., 28, 1999, pp. 155-179.

[DIN08a] THE DINI GROUP. “DN8000K10PCI User Guide”. Capturado em: http://www.dinigroup.com/DN8000k10pci.php, Novembro 2008.

[DIN08b] THE DINI GROUP. “MainBus Specification”. Capturado em: http://www.dinigroup.com/DN8000k10pci.php, Novembro 2008.

[DIN09] THE DINI GROUP. “Dini Group”. Disponível em: <http://www.dinigroup.com/>. Acesso em 15/01/09.

[DUB08] DUBEY, Rahul. “Introduction to Embedded System Design Using Field Programmable Gate Arrays”. Springer, 2008.

[ESS95] ESSMANN, U.; PERERA, L.; BERKOWITZ, M. L.; DARDEN, T.; LEE, H.; PEDERSEN, L. G. “A smooth particle mesh Ewald method”. The Journal of Chemical Physics, 103(19), 1995, pp. 8577-8593.

[EUR07] European Community Sixth Framework Programme. “JRA4: Life Sciences – Annual report and Update”. Distributed European Infrastructure for Supercomputing Applications (DEISA’07), 2007.

[GAP08] GAPH – Hardware Design Support Group. Capturado em: http://www.inf.pucrs.br/~gaph, Janeiro 2008.

[GRA82] GRAHAM, S. L.; KESSLER, P. B.; MCKUSICK, M. K. “Gprof: A call graph execution profiler”. SIGPLAN Journal, 17(6), 1982, pp. 120-126.

[GRA05] GRAHAM, R. L.; WOODALL, T. S.; SQUYRES, J. M. “Open MPI: A flexible high performance MPI”. In Fourth International Conference on Parallel Processing and Applied Mathematics (PPAM ’05), 2005.

[GRO96] GROPP, W.; LUSK, E.; DOSS, N.; SKJELLUM, A. “A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard”. Parallel Computing, 22(6), 1996, pp. 789-828.

[GRO99] GROPP, W.; LUSK, E.; THAKUR, R. “Using MPI-2: Advanced Features of the Message-Passing Interface”. The MIT Press, 1999.

[GU07] GU, Y.; HERBORDT, M. C. “High Performance Molecular Dynamics Simulations with FPGA Coprocessors”. In: Reconfigurable Systems Summer Institute (RSSI’07), 2007.

[GU08] GU, Y.; VANCOURT, T.; HERBORDT, M. C. “Explicit design of FPGA-based coprocessors for short-range force computations in molecular dynamics simulations”. Parallel Computing, 34(4-5), 2008, pp. 261-277.

[HEI05] HEIN, J.; REID, F.; SMITH, L.; BUSH, I.; GUEST, M.; SHERWOOD, P. “On the Performance of Molecular Dynamics Applications on Current High-End Systems”. Phil. Trans. Math. Phys. Eng. Sci., 363(1833), 2005, pp. 1987-1998.

[HES08] HESS, B.; KUTZNER, C.; LINDAHL, E.; VAN DER SPOEL, D. “GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation”. J. Chem. Theory Comput., 4, 2008, pp. 435-447.

[JER05] JERRAYA, A.; TENHUNEN, H.; WOLF, W. “Guest Editors’ Introduction: Multiprocessors Systems-on-Chips”. IEEE Computer, 38(7), 2005.

[KAL99] KALÉ, L.; SKEEL, R.; BHANDARKAR, M.; BRUNNER, R.; GURSOY, A.; KRAWETZ, N.; PHILLIPS, J.; SHINOZAKI, A.; VARADARAJAN, K.; SCHULTEN, K. “NAMD2: Greater Scalability for Parallel Molecular Dynamics”. Journal of Computational Physics, 151, 1999, pp. 283–312.

[KIN06] KINDRATENKO, V.; POINTER, D. “A Case Study in Porting a Production Scientific Supercomputing Application to a Reconfigurable Computer”. In: 14th IEEE Symp. Field-Programmable Custom Computing Machines (FCCM ’06), 2006, pp. 13-22.

[KUE05] KUEHN, Jeff. “A Brief Overview of Activities in the Future Technologies Group at Oak Ridge National Laboratory”. In: Workshop on Research Alliance in Math and Science (RAMS’05), 2005. Capturado em: http://www.csm.ornl.gov/workshops/ RAMSworkshop05/presentations/kuehn.pdf, Outubro 2008.

[LAB08] LABIO – Laboratório de Bioinformática Modelagem e Simulação de Biossistemas. Capturado em: http://dgp.cnpq.br/buscaoperacional/detalhegrupo. jsp?grupo=0006209DZG4E0Y, Novembro 2008.

[LAM08] LAMMPS - LAMMPS Molecular Dynamics Simulator. Capturado em: http://lammps.sandia.gov/, Novembro 2008.

[LIN01] LINDAHL, E.; HESS, B.; VAN DER SPOEL, D. “GROMACS 3.0: a package for molecular simulation and trajectory analysis”. J. Mol. Model., 7, 2001, pp. 306–317.

[LIU04] LIU, D.; DUAN, Z.; KRASNY, R.; ZHU, J. “Parallel Implementation of the Treecode Ewald Method”. In: 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 2004.

[MAR01] MARTIN, G.; CHANG, H. “Tutorial 2 - System on Chip Design”. In: 9th International Symposium on Integrated Circuits, Devices & Systems (ISIC'01), 2001, pp. 12-17.

[MAT01] MATTHEY, T.; IZAGUIRRE, J. A. “ProtoMol: A molecular dynamics framework with incremental parallelization”. In Proc. of the Tenth SIAM Conf. on Parallel Processing for Scientific Computing (PP01), 2001.

[MIL08] MILLER, B. T.; SINGH, R. P.; KLAUDA, J. B.; HODOSCEK, M.; BROOKS, B. R.; WOODCOCK, H. L. “CHARMMing: a new, flexible web portal for CHARMM”. J. Chem. Inf. Model., 48 (9), 2008, pp. 1920-1929.

[MOH09] MOHR, A.; CALAZANS, N. L. V. “Aceleração de Simulações por Dinâmica Molecular Utilizando Hardware Reconfigurável”. Plano de Estudo e Pesquisa. Porto Alegre, RS: PUCRS, Faculdade de Informática, 2009.

[MUN09] MUNDIM, K. C. “Modelagem Molecular Aplicada a Sólidos e Biomoléculas”. IV Escola de Inverno do CBPF. Capturado em: http://www.unb.br/iq/kleber/areasatuacao/EscolaCBPF/index.htm, Maio 2009.

[NAM08] NAMD – NAnoscale Molecular Dynamics. “NAMD Scalable Molecular Dynamics”. Capturado em: http://www.ks.uiuc.edu/Research/namd/, Novembro 2008.

[NAT04] NATRAJAN, A.; CROWLEY, M.; WILKINS-DIEHR, N.; HUMPHREY, M. A.; FOX, A. D.; GRIMSHAW, A. S.; BROOKS, C. L. “Studying Protein Folding on the Grid: Experiences using CHARMM on NPACI Resources under Legion”. Concurr. Comput. : Pract. Exper., 16 (4), 2004, pp. 385-397.

[NER09] NERSC – National Energy Research Scientific Computing Center. “Mixing C and Fortran on the SP”. Capturado em: http://www.nersc.gov/nusers/resources/software/ibm/c_and_f.php, Fevereiro 2009.

[ORD06] ORDONEZ, E. D. M.; PENTEADO, C. G.; SILVA, A. C. R. “Microconroladores e FPGAs: Aplicações em Automação”. Novatec Editora, 2006, pp. 275-278.

[PAT06] PATEL, A.; MADILL, C. A.; SALDAÑA, M.; COMIS, C.; POMES, R.; CHOW, P. “A Scalable FPGA-based Multiprocessor”. In: 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’06), 2006, pp. 111-120.

[PEA95] PEARLMAN, D. A.; CASE, D. A.; Caldwell, J. W.; Ross, W. S.; Cheatham III, T. E.; DeBolt, S.; Ferguson, D.; Seibel, G.; Kollman, P. “AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules”. Computer Physics Communications, 91, 1995, pp. 1-41.

[PHI05] PHILLIPS, J. C.; BRAUN, R.; WANG, W.; GUMBART, J.; TAJKHORSHID, E.; VILLA, E.; CHIPOT, C.; SKEEL, R. D.; KALÉ, L.; SCHULTEN, K. “Scalable Molecular Dynamics with NAMD”. Journal Comput. Chem., 26(16), 2005, pp. 1781–1802.

[PLI95] PLIMPTON, Steve."Fast Parallel Algorithms for Short-Range Molecular Dynamics". Journal of Computational Physics, 117, 1995, pp. 1-19.

[PLI97] PLIMPTON, S.; POLLOCK, R.; STEVENS, M. “Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations”. In Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, 1997.

[RHO08] RHOADS, S. “Plasma - most MIPS I(TM) opcodes: Overview”. Project page at Opencores.org. Capturado em: http://www.opencores.org/projects.cgi/web/mips/overview, Abril 2008.

[SAH08] SAHA, P.; EL-ARABY, E.; HUANG, M.; TAHER, M.; LOPEZ-BUEDO, S.; EL-GHAZAWI, T.; SHU, C.; GAJ, K.; MICHALSKI, A.; BUELL, D. “Portable library development for reconfigurable computing systems: A case study”. Parallel Computing, 34, 2008, pp. 245-260.

[SAL06] SALDAÑA, M.; CHOW, P. “TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs”. In: International Conference on Field Programmable Logic and Application (FPL’06), 2006, pp. 1-6.

[SCH05] SCHROEDER, E. K.; BASSO, L. A.; SANTOS, D. S.; SOUZA, O. N. “Molecular dynamics simulation studies of the wild-type, I21V and I16T mutants of isoniazid resistant Mycobacterium tuberculosis enoyl reductase (InhA) in complex with NADH: Towards the understanding of NADH-InhA different affinities”. Biophysical Journal, 89(2), 2005, pp. 876-884.

[SCR08] SCROFANO, R.; GOKHALE, M. B.; TROUW, F.; PRASANNA, V. K. “Accelerating Molecular Dynamics Simulations with Reconfigurable Computers”. In: IEEE Transactions on Parallel and Distributed Systems, 19(6), 2008, pp. 764-778.

[SMI96] SMITH, W.; FORESTER, T. “DL_POLY: a macromolecular simulation package”. J. Mol. Graph., 14, 1996, pp. 136.

[STA07] STAHLBERG, E.; POPIG, D.; RYLE, D.; STEINKE, T.; BABST, M.; TAHER, M.; ANDERSON, K. “Molecular Simulations with Hardware Accelerators: A Portable Interface Definition for FPGA Supported Acceleration”. In: Reconfigurable Systems Summer Institute (RSSI’07), National Center for Supercomputing Applications, 2007.

[STO09] STOTE, R.; DEJAEGERE, A.; KUZNETSOV, D.; FALQUE, L. “CHARMM Tutorial: Molecular Dynamics Simulation”. Capturado em: http://www.ch.embnet.org/MD_tutorial, Maio 2009.

[STR08] STRENSKI, Dave. “FPGA Floating Point Performance – a pencil and paper evaluation”. HPC Wire, 2007. Capturado em: http://www.hpcwire.com/ hpc/1195762.html, Outubro 2008.

[SUT02] SUTMANN, Godehard. “Classical Molecular Dynamics”. John von Neumann Institute for Computing, 10, 2002, pp. 211-254.

[TAU02] TAUFER, M.; PERATHONER, E.; CAVALLI, A.; CAFLISCH, A.; STRICHER, T. “Performance Characterization of a Molecular Dynamics Code on PC Clusters Is there any easy parallelism in CHARMM?”. In: International Parallel and Distributed Processing Symposium (IPDPS’02), 2002, pp. 15-19.

[TCB08] TCBG – Theoretical and Computational Biophysics Group. “VMD – Visual Molecular Dynamics”. Capturado em: http://www.ks.uiuc.edu/Research/vmd/, Novembro 2008.

[TOD04] TODOROV, I.; SMITH, W. “DL_POLY_3: the CCP5 national UK code for molecular dynamics simulations”. Phil. Trans. Math. Phys. Eng. Sci., 362(1822), 2004, pp. 1835-1852.

[UND06] UNDERWOOD, K. D., HEMMERT, K. S., ULMER, C. “Architectures and APIs: Assessing Requirements for Delivering FPGA Performance to Applications”. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC’06), 2006.

[WIL06] WILLIAMS, J. A.; SYED, I.; WU, J.; BERGMANN, N. W. “A Reconfigurable Cluster-on-Chip Architecture with MPI Comunication Layer”. In: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’06), 2006, pp. 350-352.

[YAN07] YANG, X.; MOU, S.; DOU, Y. “FPGA-Accelerated Molecular Dynamics Simulations: An Overview”. In: International Workshop on Applied Reconfigurable Computing (ARC’07), 2007, pp. 293-301.

APÊNDICE A – PASSOS PARA UTILIZAÇÃO DA API

A arquitetura de software construída foi validada de maneira específica dentro do projeto proposto. Essa validação deu-se com a utilização do software PMEMD, como aplicação de simulação por dinâmica molecular, presente na máquina hospedeira e com a comunicação através da interface PCI com a plataforma FPGA, modelo DN8000K10PCI.

Para poder executar as chamadas da API e efetuar a comunicação entre a máquina hospedeira e a plataforma de hardware é preciso seguir os seguintes passos:

1. Alterar os arquivos de configuração do PMEMD
2. Adicionar arquivos da API
3. Adicionar chamadas no código Fortran do PMEMD
4. Compilação do PMEMD
5. Preparar a plataforma de hardware
6. Executar a Simulação por dinâmica molecular

Os passos serão descritos conforme recursos de hardware e software, com seus devidos modelos e versões, discutidos nos Capítulo 4 e 6.

1. Alterar os arquivos de configuração do PMEMD

Nesse primeiro passo é necessária a alteração de dois arquivos de configuração do software PMEMD. Quando for realizada uma instalação típica com esse software o primeiro comando a ser executado é para gerar o arquivo de configuração config.h, esse comando (./configure) precisa ser executado com três argumentos, a arquitetura do processador, o compilador Fortran e o tipo de instalação (Serial ou Paralela), isso pode ser visto com maiores detalhes no "readme" que acompanha o software. Nesse trabalho, escolheu-se a instalação serial, pois o trabalho corresponde ahá apenas um nodo (máquina hospedeira).

No arquivo de configuração config.h é necessário adicionar duas linhas e alterar outras duas. As duas linhas, abaixo , devem ser inseridas no início do arquivo, correspondem, respectivamente, a determinar o sistema operacional base e o diretório onde se encontra o driver da plataforma de hardware.

```
DESTOS = LINUX
DRIVER_DIR = ../linuxdrv-2.6
```

E as duas linhas que devem ser alteradas estão descritas a seguir, para adicionar os *flags* de compilação e a biblioteca lcurses.

```
CFLAGS = -O2 -Wall -D$(DESTOS) -I$(DRIVER_DIR) -c -o $@
LOADLIBS = -lcurses -limf -lsvml -Wl,-rpath=$(IFORT_RPATH)
```

O outro arquivo de configuração que precisa ser modificado é o *Makefile* do seguinte caminho *../amber9/src/pmemd/src.* Nesse arquivo é necessário adicionar uma linha na lista de arquivos objetos com os seguintes nomes.

```
OBJS =
          os_dep.o pci.o dn8k10pci.o\
```

Outra alteração a ser realizada é na linha de sufixos de arquivos fontes e objetos.

```
.SUFFIXES: .fpp .c .o .obj
```

A última e maior modificação nesse arquivo é a remoção das duas linhas a seguir e a adição de oito linhas na compilação dos novos arquivos fontes e do arquivo modificado pmemd_clib.c.

```
.c.o:
    $(CC) $(CFLAGS) -c $*.c
```

As oito linhas a serem adicionadas no arquivo *Makefile*.

```
os_dep.o: pci.h os_dep.h os_dep.c
    $(CC) $(CFLAGS) os_dep.c
pci.o: os_dep.h pci.h pci.c
    $(CC) $(CFLAGS) pci.c
dn8k10pci.o: os_dep.h pci.h dn8k10pci.c
    $(CC) $(CFLAGS) dn8k10pci.c
pmemd_clib.o: os_dep.h dn8k10pci.h pmemd_clib.c
    $(CC) $(CFLAGS) pmemd_clib.c
```

Após a modificação desses arquivos de configuração conclui-se o primeiro passo para a utilização da API.

2. Adicionar arquivos da API

Os arquivos a serem adicionados junto aos fontes do software PMEMD são 8 (oito): dn8k10pci.c, dn8k10pci.h, os_dep.c, os_dep.h, pci.c, pci.h, pmemd_clib.c e pmemd_clib.h. O caminho onde serão inseridos esses arquivos é o *../amber9/src/pmemd/src.*

Outro conjunto de arquivos que são necessários para inclusão da API, são aqueles referentes ao driver da plataforma de hardware. O driver adicionado é destinado ao sistema operacional linux com kernel posterior ao 2.6, e os arquivos devem ser colocados no seguinte diretório *../amber9/src/pmemd/linuxdrv-2.6/,* finalizando o segundo passo.

3. Adicionar chamadas no código Fortran do PMEMD

As chamadas da API podem ser adicionadas em qualquer módulo pertencente ao PMEMD, desde que, respeite as primitivas definidas no Capítulo 7. Um exemplo demonstrado nas linhas abaixo é a inserção da chamada para enviar uma variável do tipo inteira para a plataforma FPGA, que está contida dentro da função *short_ene_vec*. A primeira linha corresponde a chamada à API e as outras a um trecho de código em Fortran.

```
call api_data_i(CONT,nxt,1,0)
do nxt_idx = 1, nxt_cnt
 vec_idx = nxt(nxt_idx)
 delr2 = delr2_vec(vec_idx)
```

As chamadas da API podem ser adicionadas em qualquer parte do código, respeitando o padrão definido e precedido do comando *call*.

4. Compilação do PMEMD

No caminho *../amber9/src/pmemd/*, a compilação do software PMEMD é feita da mesma forma, como senão existisse a API. É preciso executar apenas dois comandos:

```
make clean
make install
```

5. Preparar a plataforma de hardware

Se o passo anterior completou-se com sucesso está pronta a inserção da API. E nesse momento é só carregar o arquivo .bit para o FPGA para poder fazer a simulação utilizando a API para a comunicação. Uma maneira simples de carregar o arquivo .bit para o FPGA é através de linha de comando com o software *Impact* da Xilinx, utilizando o modo batch desse software. As linhas a seguir mostram cada linha de comando para tal finalidade, há a necessidade do arquivo .bit estar no mesmo local do início das execuções dos comandos.

```
impact -batch

setMode -bscan
setCable -p auto
identify
addDevice -p 1 -file arquivo.bit
program -p 1
quit
```

6. Executar a Simulação por dinâmica molecular

O último passo é a execução da simulação por dinâmica molecular, que com a inclusão da API não é necessária fazer nenhuma modificação nessa execução, tornando oculto ao usuário final essa adaptação. Assim, uma execução padrão utilizada nesse trabalho foi a com o comando a seguir. Os três primeiros arquivos são os de entrada para a simulação e os três últimos são os arquivos de saída gerados pelo software PMEMD, para execução de posteriores simulações e com tempos de execução da simulação.

```
pmemd -O -i arquivo_DM.in \
-c arquivo_c_vel.cav \
-p arquivo_est_molec.top \
-o pmemd_saida.out \
```

```
-r pmemd_saida.cav \
-x pmemd_saida.crd
```

Desta forma, conclui-se esse breve tutorial de utilização da API para a comunicação entre máquina hospedeira e a plataforma FPGA.

ANEXO A – CARACTERÍSTICAS DA DINÂMICA MOLECULAR

O arquivo de entrada, Figura 39, correspondente aos parâmetros de dinâmica molecular utilizados em todas as simulações nos estudos de casos e experimentos demonstrados nesse trabalho, esse arquivo foi criado pelo grupo [LAB08].

```
&cntrl
 imin=0, ntx=5, irest=1, ntrx=1,
 ntxo=1, ntpr=250,
 ntwr=250, iwrap=0, ntwx=250, ntwv=0, ntwe=0,
 ioutfm=0, ntwprt=0, idecomp=0,
 ibelly=0, ntr=0,
 nstlim=500, nscm=1000, t=0.0, dt=0.002, nrespa=1,
 tsgavg=0.2, tempsg=1.0, sgft=0.0,
 ntt=1,temp0=298.16,   tempi=298.16,   ig=71277,   tautp=1.0,   gamma_ln=0,
vrand=0, vlimit=20.0,
 ntp=1, pres0=1.0, comp=44.6, taup=2.0,
 ntc=2, tol=0.00001,jfastw=0,
 ivcap=0,
 ntf=2,  ntb=2,  dielc=1.0,  cut=9.0,  scnb=2.0,  scee=1.2,  nsnb=25,  ipol=0,
ifqnt=0,
 igb=0, ievb=0,
 /
 &ewald
 verbose=0,   ew_type=0,   nbflag=1,   use_pme=1,   vdwmeth=1,   eedmeth=1,
netfrc=0,
 /
```

Figura 39 - Arquivo de entrada para as simulações por DM.

No pacote AMBER, existe uma grande variedade de parâmetros a serem utilizados como entrada para a simulação e diversas variações em cada um desses parâmetros. Na Tabela 7, foram descritos apenas os parâmetros apresentados no arquivo de entrada das simulações, conforme Figura 39, definindo brevemente cada um desses parâmetros para ajudar na compreenção dos conceitos relacionadas à dinâmica molecular.

Tabela 7 – Definição dos parâmetros da dinâmica molecular utilizada nas simulações [AMB08a].

Parâmetros	Descrição
imin	Flag para executar minimização. Se igual a 0, sem minimização (somente faz dinâmica molecular; padrão).
ntx	Opção para leitura de coordenadas iniciais, velocidades e tamanhos de caixas do arquivo "inpcrd". Se igual a 5, X e V são lidos formatados; informações da caixa podem ser lidas se ntb>0. A informação de velocidade somente é usada se irest=1.
irest	Flag para reiniciar a execução. Se igual a 1, reinicia os cálculos. Necessita de velocidades em arquivos de coordenadas de entrada, assim você pode reiniciar NTX sem reiniciar a DM.
ntrx	Formato de coordenadas cartesianas para restrições do arquivo "refc". Se igual a 1, está formatado como padrão ASCII.
ntxo	Formato das coordenadas finais, velocidades, e tamanhos de caixa (se executam volume ou pressão constantes) escritas para o arquivo "restrt". Se igual a 1, está formatado como padrão.
ntpr	Em cada NTPR de passos as informações de energias serão impressas para os arquivos

	"mdout" e "mdinfo". "mdinfo" é fechado e reaberto a cada vez, desta forma sempre contem a mais recente energia e temperatura, o Padrão é 50.
ntwr	Em cada NTWR de passos durante a dinâmica, será escrita para o arquivo "restrt", garantindo a recuperação de uma colisão que não seja difícil.
iwrap	Se marcado com 1, as coordenadas escritas para reiniciar e arrumar os arquivos de trajetórias dentro da caixa primária. O padrão (quando iwrap=0) está para não apresentar alguma manipulação.
ntwx	Em cada NTWX de passos as coordenadas será escrita para o arquivo "mdcrd" na forma compacta. O padrão é NTWX=0, inibe todas as saídas.
ntwv	Em cada NTWV de passos a velocidade será escrita para o arquivo "mdvel" na forma compacta. O padrão é NTWV=0, inibe todas as saídas.
ntwe	Em cada NTWE de passos a energia e temperatura irão ser escritas para o arquivo "mden" na forma compacta. O padrão é NTWE=0, inibe todas as saídas.
ioutfm	Formato de coordenadas e velocidades marcadas, se igual a 0 está formatado como padrão.
ntwprt	Flag para limitar arquivos de coordenadas e velocidades, este flag é usado para diminuir os arquivos de coordenadas e velocidades somente incluindo porções do sistema de grande interesse. Se igual a 0, inclui todos os átomos no sistema.
idecomp	Esta opção na realidade somente é usada em conjunto com mm_pbsa, se igual a 0, não faz nada(Padrão).
ibelly	Flag para a dinâmica do tipo Belly, se igual a 0, não executa Belly(padrão).
ntr	Flag para restringir específicos átomos no espaço cartesiano usando um potencial harmônico, se igual 0, não tem restrições posicionais.
bellymask	Cadeia de caracteres que especifica movimento dos átomos quando ibelly=1. A sintaxe para ambas restraintmask e bellymask está descritas na Seção 13.5 do arquivo [AMB08a]. Note que estas cadeias de caracteres mascarados são limitados para um máximo de 256 caracteres.
restraint_wt	O peso (em kcal/mol − Å2) para restrições posicionais.
restraintmask	Cadeia de caracteres que especifica restrições a átomos quando ntr=1, neste caso está sem restrições.
nstlim	Número de passos de MD a serem executados, o padrão é 1.
nscm	Flag para remoção de translação e rotação do movimento de centro de massa para intervalos regulares, o padrão é 1000.
t	Tempo de início (ps), este é para sua própria referência e não é crítico.
dt	É o tempo dos passos (ps), o padrão é 0.001.
nrespa	Esta variável permite ao usuário avaliar condições de baixa variação nos campos de força menos freqüentes, para o PME é uma ferramenta para a soma recíproca.
isgld	Com o valor zero desabilita o self-guiding.
tsgavg	Tempo médio local (psec) para o cálculo de força direcional, padrão 0.2.
tempsg	Direciona a temperatura, em conjunto com o sgft definem a intensidade de força em unidades de temperatura, o padrão é 1.0.
sgft	Fator de direção, define a intensidade de força quando tempsg=0 e se tempsg>0, impõem-se o valor de sgft, porque sgft varia com o sistema e condições de simulação.
isgsta	O índice do primeiro átomo da região SGLD.
isgend	O índice do ultimo átomo da região SGLD (Self-guided Langevin dynamics). A SGLD pode ser usado para se adaptar buscando eficiência na simulação por DM (neste caso pois gamma_ln=0)
ntt	Regulação de temperatura, se igual a 1, utiliza o algoritmo weak-coupling um fator simples de escala é usado em todos os átomos.
temp0	Referente a temperatura em que o sistema deve manter, se ntt>0. Padrão 300.
tempi	Temperatura inicial.
ig	É a origem para o gerador de números aleatórios, a velocidade inicial da MD é dependente deste quando NTX=3 e TEMPI=0.0, e o padrão é 71277.
tautp	Constante de tempo, para aquecer o banho acoplado ao sistema, o padrão é 1.0.

gamma_ln	Frequência de colisão, o padrão é 0.
vrand	Se vrand>0 e ntt=2 as velocidades podem ser aleatórias conforme a temperatura, não é o caso aqui.
vlimit	Se algum componente tem uma velocidade alta, este parâmetro irá reduzir para o número definido nele.
ntp	Flag para constante de pressão dinâmica. Esta opção tem que ser definida como 1 ou 2 Constante Pressão, quando o limite periódico condicional é usado (NTB = 2). Se igual a 1 tem DM com escala isotrópica de posição.
pres0	Pressão de referência, o padrão é 1.0.
comp	Compressibilidade do sistema quando NTP > 0. Um valor de 44.6 (padrão) é apropriado para água.
taup	Controle de Pressão.
ntc	NTC ≈ NTF
tol	Tolerância relativa geométrica para reiniciar as coordenadas, máximo recomendado : <0.00005 Å; padrão 0.00001.
jfastw	Flag para definir água consistente, se igual a 0, é uma operação normal e as águas são identificadas por nomes padronizados.
ivcap	Flag de Controle para Water Cap, se igual a 0, pode ter efeito se estiver no arquivo prmtop (padrão)
ntf	Avaliação de Força, quando ntf = 2, omissão de interações ligantes envolvendo H-atomos (usado com NTC=2).
ntb	Limite Periódico, os valores de NTB especificam constante de volume ou contante pressão dinâmica podem ser usados, com ntb = 2 é usado para constante de pressão. para sistemas periódicos, pressão constante é a maneira utilizada para equilibrar a densidade e iniciar um estado, não é preciso.
dielc	Uma constante multiplicativa dielétrica para interações eletrostáticas, o padrão é 1.0 e não está relacionado a simulações Generalized Born.
cut	É usado para especificar o raio de corte de átomos não ligados em Å. Para o PME este corte é utilizado para limitar diretamente a extensão da soma, um bom valor utilizado por padrão é o 8.0 Å para o raio de corte de átomos não ligados.
scnb	Parâmetro de controle dos campos de força. 1-4 é o fator de escala de átomos não ligados(divisão de interações de VDW); O padrão é 2.0.
scee	Parâmetro de controle dos campos de força. 1-4 é o fator de escala eletrostática (divisão de interações eletrostáticas); O padrão é 1.2.
nsnb	Determina a frequência da lista de atualizações de não ligados quando igb=0 e nbflag=0; O padrão é 25.
ipol	Campo de Força Polarizado. Quando estiver com o valor 1, utiliza este campo de força polarizável, mas o padrão é 0.
ifqnt	Cálculos de QM/MM. Se estiver com o valor 1, necessita preparar uma lista de nomes &qmmm com parâmetros adicionais, mas o padrão é 0.
igb	GB (Generalized Born). Flag para usar modelos de solventes implícitos o Generalized Born or Poisson-Boltzmann, com o valor em 0, não é utilizado este modelo.
ievb	Campo de Força EVB (Empirical Valence Bond). Se estiver com o valor 1, usa a ligação de valência empírica para computar energias e forças, mas o padrão é 0.
iamoeba	Campo de Força AMOEBA. Se estiver com o valor 1, necessita preparar uma lista de nomes amoeba com parâmetros adicionais, mas o padrão é 0.
verbose	O padrão é 0, mas se for configurado com valores altos (máximo de 3) gera volumosas informações de saídas na execução do PME
ew_type	O padrão usado é 0 no qual é configurado o método PME. Se fro igual a 1, por exemplo, ao invés de aproximar, executa o PME interpolado com um cálculo regular de Ewald.
nbflag	Se igual a 0, constrói a soma direta da lista sem-ligação no caminho passado e atualiza esta lista em cada passo do nsnb.
use_pme	Para usar o método PME.

vdwmeth	Determina o método usado para interações de Van der Waals incluindo depois uma soma direta. Um valor 0 inclui sem correções; O valor padrão usado é 1, um contínuo modelo de correção para energia e pressão.
eedmeth	Determina como alternar uma função para somas diretas em interações de Coulomb avaliadas. O valor padrão é 1 usa-se um cubic spline
netfrc	A básica implementação PME "smooth" não usada aqui necessariamente para conservar o momento. Este parâmetro é marcado em 0 se minimização é requisitada, no qual implica que o gradiente é uma precisa derivada de energia.