

Técnicas de paralelismo multiníveis: uma abordagem aplicada a bioinformática

Felipe Fernandes Albrecht

¹ Instituto Militar de Engenharia (IME) – Rio de Janeiro, RJ – Brasil

felipe.albrecht@gmail.com

Resumo. Com o crescimento das informações genéticas moleculares no contexto computacional na ordem exponencial, cada vez mais são necessários grupos computacionais de alto desempenho para lidar com tais dados. Porém, os dados crescem em ordem mais elevada que os avanços nos processadores atuais. Para lidar com este dilema, este trabalho apresenta alguns níveis de paralelismo na arquitetura de processadores mais utilizados para tais tarefas, desde instruções com múltiplos dados até grids computacionais, com o objetivo de diminuir a distância do poder computacional da quantidade de dados disponíveis. Por fim, são apresentadas sugestões de otimizações aplicáveis a maioria dos softwares de bioinformática.

1. Introdução

O crescimento de informações genéticas moleculares no contexto computacional cresce numa ordem exponencial. Na figura 1 pode-se observar o crescimento da quantidade de bases seqüenciadas depositadas no *GenBank* (<http://www.ncbi.nlm.nih.gov/Genbank/>). As sequências depositadas neste banco de dados primário servem como dados para outros experimentos, e estes experimentos geram mais dados que devem ser armazenados e analisados.

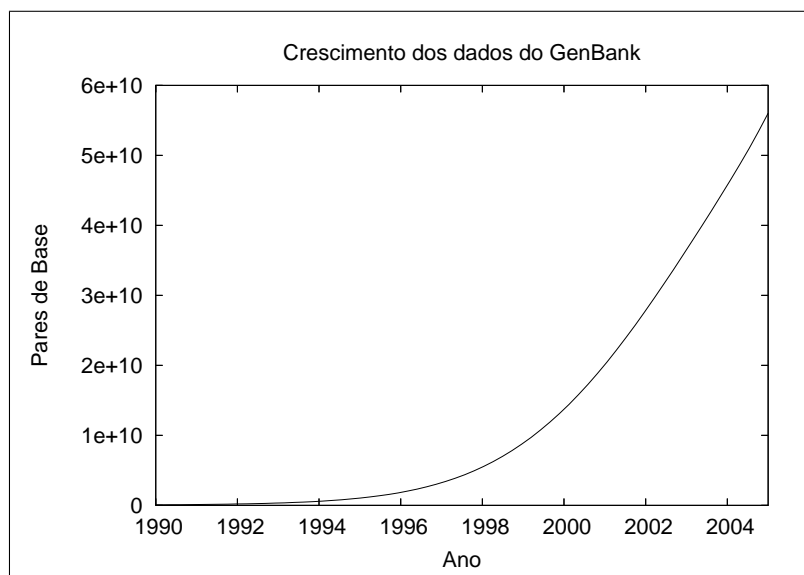


Figura 1. Crescimento da quantidade bases seqüenciadas depositadas no GenBank.

Os sistemas computacionais que analisam tais informações são computadores que seguem a arquitetura *von Neuman*. A arquitetura *von Neuman* define um computador de

propósito geral, onde possui uma unidade central de processamento, a *central processing unit (CPU)* ou processador, uma memória primária, de onde os programas e dados são lidos e armazenados durante a execução dos programas e por fim, uma unidade de entrada e saída.

Em 1965, Gordon Moore previu que o número de transistores num chip dobra em média a cada dois anos (INTEL, 2007b). Este aumento na quantidade de transistores representa de forma geral, para o usuário final, um aumento no desempenho dos processadores e um aumento na capacidade das memórias primárias. Dando um exemplo, Hennessy e Patterson (2003, pg. 283) informam que o desempenho dos microprocessadores aumentou na taxa de 55% ao ano a partir de 1987.

Mesmo com a taxa de crescimento de 55% ao ano, o desempenho dos processadores cresce de forma menor que o crescimento da quantidade de informações bio-moleculares disponíveis em banco de dados públicos. Isto significa que o tempo de processamento das informações contidas nestes bancos tende a aumentar, mesmo obtendo-se os processadores mais modernos. O problema se repete com outro recurso fundamental para o processamento destas informações: a memória principal, ou memória *RAM* (Random Access Memory). Sendo que o custo da memória decai de forma inferior ao crescimento das informações bio-moleculares disponíveis nos bancos de dados.

O aumento do desempenho dos processadores é fator fundamental para os futuros projetos de pesquisa que utilizam dados bio-moleculares. Para validar tal assertiva, basta analisar os principais projetos de seqüenciamento de genomas, onde a quantidade de cresce e para tornar viável a análise de futuros dados, confia-se no aperfeiçoamento do desempenho dos computadores como um todo e principalmente nos processadores. Porém, nas últimas décadas, o aumento do desempenho dos processadores esteve fortemente atrelada ao aumento do “*clock*”, ou de outra forma, ao aumento da quantidade de instruções executadas por segundo (*Mhz*) nos processadores.

Na metade da primeira década do século 21 a chamada “guerra dos *Mhz*” chegou ao fim (HEBENSTREIT, 2007). Isto significa que ao invés de buscar meios de incrementar a quantidade de *Mhz* de um processador, começou-se a buscar outros meios para aperfeiçoar o uso destes ciclos e o uso de técnicas de paralelismo. Entre estes meios pode-se citar: a execução de múltiplas instruções no mesmo ciclo, através de *pipelines* mais eficientes e de tecnologias de *hiperthreading*, a execução de instruções que executam operações em múltiplos dados e o desenvolvimento de processadores com múltiplos núcleos. Em outras palavras, significa que ao invés de aperfeiçoar o desempenho dos processadores na execução sequencial, busca-se meios de otimizar as execuções paralelizando as execuções dos *softwares* em diversos níveis.

Desta forma, este artigo explora alguns níveis de paralelismo disponíveis na arquitetura *AI-32* e *IA-64* e propor otimizações nos algoritmos e nas execuções dos softwares de bioinformática para obter-se vantagens no uso destas novas tecnologias. Também é visto a utilização de *cluster* computacionais e sua utilização com outros níveis de paralelismo.

2. Níveis de paralelismo

Paralelismo no sentido computacional, significa executar paralelamente trechos de programas, programas ou até conjuntos de programas simultaneamente com dois objetivos:

aumentar a confiabilidade ou o desempenho do sistema computacional. Este artigo tem como objetivo utilizar o paralelismo para o aumento do desempenho computacional.

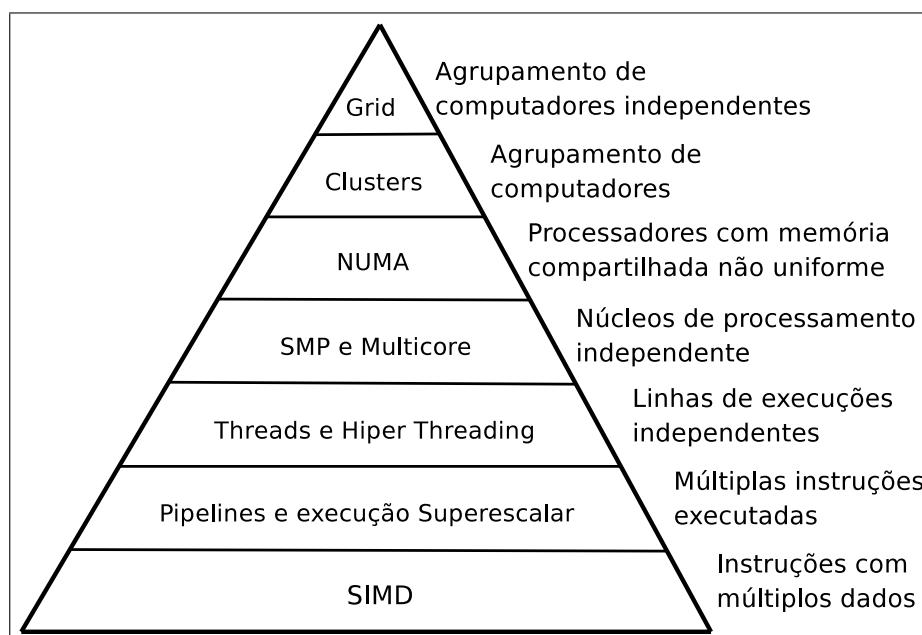


Figura 2. Níveis de paralelismo ordenados por granularidade.

O paralelismo podem ser dividido em alguns níveis, variando principalmente na granularidade dos dados tratados. A granularidade é o tamanho dos dados: variando de *bytes* significando variáveis no programa a *gigabytes* contendo informações referentes a um fluxo de um *workflow* científico. Dependendo da granularidade dos dados, um nível de paralelismo é utilizado, variando de dados numa instrução, paralelismo de execução de instruções, de linhas de execução do *software* e em múltiplas execuções do mesmo *software*. Na figura 2 é apresentado uma pirâmide onde busca-se representar a granularidade das informações paralelizadas. Em cada nível está escrito o nome da tecnologia e ao seu lado uma breve descrição do seu funcionamento. Ao longo deste artigo, cada nível é descrito e melhorias aplicáveis à bioinformática são apresentadas, por fim, é sugerido como as paralelizações poderiam ser sobrepostas para obter um desempenho superior nas aplicações de bioinformática.

2.1. Instruções com múltiplos dados

O nível de paralelismo com menor granularidade são as instruções com múltiplos dados (*SIMD*, *single-instruction, multiple-data*). As instruções *SIMD* foram introduzidas na arquitetura *IA-32* com a tecnologia *MMX* com o objetivo de acelerar o desempenho de mídias avançadas e aplicativos de comunicação (INTEL, 2005). A tecnologia *MultiMedia eXtensions (MMX)* define um conjunto de novos registradores e instruções para o processador. Na figura 3 é apresentado o funcionamento de uma instrução com múltiplos dados. As duas fontes são dois registradores, cada qual contendo 4 dados e o destino, é outro registrador na qual armazenará o resultado da operação executada. As operações podem ser aritméticas, comparações, conversões e empacotamento de dados. Observando a figura 3 nota-se a vantagem do uso de instruções *SIMD*, onde em uma instrução, 4 operações são executadas simultaneamente.

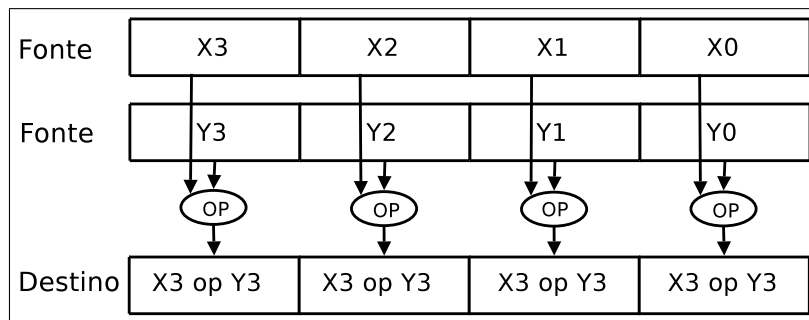


Figura 3. Exemplo de execução SIMD.

Após o *MMX*, outras extensões de instruções *SIMD* foram lançadas para os processadores *IA-32*. Estas extensões são chamadas de *Streaming SIMD Extensions (SSE)* e basicamente compreendem de um novo conjunto de instruções e novos registradores de dados e um de controle. Até metade do ano 2007, foram lançadas aperfeiçoamentos na extensão *SSE*, existindo desde a *SSE 1* até a *SSE 3*, sendo que o quarto lançamento esta previsto para o quarto trimestre de 2007 (RAMANATHAN, 2007).

As instruções com múltiplos dados são úteis na bioinformática para os cálculos com matrizes de distância para o alinhamento de seqüências. Rognes e Seeberg (2000) apresentam uma modificação do algoritmo *Smith-Waterman* utilizando as extensões *MMX* e *SIMD* para o cálculo das matrizes de distâncias neste algoritmo. Neste trabalho é relatado um ganho de aproximadamente seis vezes a outras implementações que não utilizam *SIMD*. Uma possível otimização no *BLAST* (ALTSCHUL et al., 1997) é utilizar as instruções *SIMD* disponíveis na arquitetura *IA-32*.

É importante salientar que as extensões *MMX* e *SSE* estão presentes em todos os processadores atuais da arquitetura *IA-32*, tanto dos fabricantes *Intel* como da *AMD*.

2.2. Paralelismo em nível de instrução

Todos os computadores desde 1985 aproximadamente utilizam *pipelines* para sobrepor a execução de instruções e melhorar o desempenho (HENNESSY; PATTERSON, 2003, pg. 126). *Pipelining* é uma técnica de implementações de processadores pela qual várias instruções são sobrepostas na execução; ela tira proveito do paralelismo que existe entre ações necessárias para executar uma instrução. Hoje o *pipelining* é uma técnica-chave de implementação utilizada para tornar as *CPUs* mais rápidas (HENNESSY; PATTERSON, 2003, pag. 656). Hennessy e Patterson (2003) descreve um pipeline é semelhante a uma linha de montagem em que cada etapa opera em paralelo com as outras. Na execução de instruções num processador, cada instrução a ser executada possui diversas fases, como o carregamento da instrução, decodificação da instrução, execução, acesso a memória e escrita nos registradores. O objetivo do uso de *pipelines* é que enquanto uma instrução está numa fase, pode-se executar outra instrução numa outra fase.

Para utilizar os benefícios da tecnologia de *pipeline* o desenvolvedor deve preocupar-se em desenvolver *softwares* em que as variáveis estejam fracamente acopladas, minimizar o uso de desvios e também em conhecer o compilador e suas opções de otimização para que o código de binário gerado dê ao processador a capacidade de utilizar o seu *pipeline*. Desta forma, *softwares* com alta carga de processamento, como os de cálculo ma-

temático para construção de modelos estatísticos de famílias de proteínas ou simulações de acoplamento de proteínas e enzimas, devem ser compilados para que se use as otimizações disponíveis no processador que irá executar tais *softwares*.

2.3. Threads

As *threads* são linhas de execução independentes de um *software* e são normalmente utilizadas em interfaces do usuário para simular a execução simultânea de partes do programa. Exemplificando, quando um determinado *software* está abrindo um arquivo e o usuário pode executar outras operações neste *software*. O uso de *threads* na bioinformática é interessante quando se faz o uso de serviços *web* e leitura e escrita de grandes arquivos, pois enquanto o sistema está esperando o retorno de uma solicitação de leitura ou escrita, o processador pode continuar a trabalhar em outros cálculos no mesmo *software*.

Um aperfeiçoamento nas tecnologias de *pipelines* é a tecnologia *Hyper-Threading*. Esta tecnologia permite um único processador físico executar duas ou mais linhas de execuções (*threads*) independentes concorrentemente usando recursos de execução compartilhados (INTEL, 2005, pg. 2-13b). Desta forma, o *software* enxerga dois processadores separados, enquanto fisicamente existe somente um. Porém, para obter as vantagens do uso desta tecnologia, o *software* deve estar utilizando *threads* ou de mais de um *software* com alta carga de processamento deve estar sendo executado no processador.

2.4. Multiprocessadores

Os multiprocessadores são sistemas computacionais com mais de um processador onde a memória é compartilhada entre todos os processadores num único espaço de endereçamento e a comunicação é feita por intermédio de variáveis compartilhadas armazenadas na memória. Segundo Hennessy e Patterson (1998, pg. 415) Existem dois estilos diferentes de multiprocessadores com um único espaço de endereçamento. O primeiro gasta o mesmo tempo para acessar a memória principal, não importando qual dos processadores requisitou o acesso, nem o tipo de trabalho requisitado. Tais máquinas são conhecidas como multiprocessadores *UMA* (*uniform memory access*) ou multiprocessadores *SMP*. No segundo estilo, alguns acessos à memória são mais rápidos que os outros, dependendo de qual dos processadores solicita acesso a qual endereço. Tais máquinas são conhecidas como multiprocessadores *NUMA* (*nonuniform memory access*).

Como dito anteriormente, multiprocessadores *SMP* são computadores que possuem mais de um processador e que se comunicam através de memória compartilhada de acesso uniforme. Uma classe recente de multiprocessadores *SMP* são os processadores multicóres. Estes processadores possuem mais de uma unidade de processamento no mesmo *chip* e estão se tornando padrão no mercado de novos processadores. Ao invés das indústrias investirem no aumento de ciclos por segundo, estão investindo na quantidade de núcleos por *chip*. Desta forma, os *softwares* de bioinformática devem estar preparados para a mudança de paradigma e deve-se começar a paralelizar suas execuções.

O software Blast distribuído pelo *NCBI* em <ftp://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST-BLAST/> oferece a opção de especificar em quantos processadores ele será executado. Isto é importante, pois normalmente os bancos de dados do *BLAST* são maiores que a memória primária disponível nos computadores. Então, ao invés de fazer n execuções e a necessidade de trazer para a memória a base de dados n

vezes, faz-se uma única execução com diversas linhas de execução, cada qual com uma seqüência a ser pesquisada. Desta forma, cada trechos do banco é carregado uma única vez a memória e a execução se fará mais rápida.

Ainda é difícil encontrar *softwares* de bioinformática que fazem proveito do uso de múltiplos núcleos de processamento. Nestes casos, a melhor solução para casos onde há uma grande quantidade de dados que devem ser processados é aumentar a granularidade dos dados. Desta forma, executa-se diversas instâncias do *software*, cada qual alocada a um processador e com uma parte desta massa de dados, este tipo de paralelismo é chamado de *MIMD (multiple-instruction, multiple-data)*. Técnica semelhante é utilizada em *grids* computacionais.

Outras questão na bioinformática, é que as linguagens onde os *scripts* são escritos, principalmente *PERL*, não possuem meios de paralelizar a execução do *script*. Um solução pode ser o uso de *threads*, porém um padrão para o desenvolvimento de *softwares* paralelizados em ambiente de memória compartilhada esta sendo adotado, este padrão, chamado de *OpenMP (OPENMP, 2007)*. Andrews (2000, pag. 595) descreve o *OpenMP* como um conjunto de diretivas de compilador e bibliotecas de rotinas que são usadas para expressar paralelismo em ambientes de memória compartilhada. A sua *API* foi desenvolvida por um grupo representando os principais fornecedores de *software* e *hardware* de alto desempenho. Atualmente ele está disponível para as linguagens *Fortran, C* e *C++*. O *OpenMP* foi incorporado na versão 4.1 da coleção de compiladores *GCC (FOUNDATION, 2006)* através do projeto *GOMP (FOUNDATION, 2007)*. Sendo um padrão não exclusivo a uma linguagens de programação, o *OpenMP* pode ser implementado em outras linguagens, inclusive a linguagem *PERL*, assim, facilitando a utilização de paralelismos e conseqüentemente a utilização da capacidade computacional disponível em multiprocessadores e processadores *multicore*. Uma outra biblioteca com o objetivo de facilitar o desenvolvimento de aplicativos para processadores *multicore* é a *Threading Building Blocks* desenvolvida pela *Intel* e que recentemente foi liberada com a licença *GPL*, o que tornou-a um *software* livre. Ela tem o objetivo de permitir a programadores das linguagens *C* e *C++* desenvolverem aplicativos paralelos sem o conhecimento prévio de *threads* e *paralelismo* (REINDERS, 2007).

Os multi-computadores *NUMA* são sistemas computacionais com a quantidade de processadores normalmente superior aos sistemas *SMP* e também são mais facilmente escaláveis, porém, segundo Hennessy e Patterson (1998) do ponto de vista de programação, é mais difícil obter uma melhor performance em máquinas *NUMA* do que numa *UMA*. Nota-se que dentro de um sistema *NUMA* é comum existirem sub-sistemas *SMP*, com mais de um processador na mesma placa e/ou processadores multicore. Para aplicações de bioinformática, os problemas na utilização de sistemas *NUMA* são similares aos das máquinas *SMP*, sendo que a questão dos tempos de acesso diferentes a memória é administrada pelo sistema operacional.

3. Clusters

Cluster é um termo largamente utilizado e significa uma interligação de computadores através de software e rede independentes num único sistema, ou seja, uma interligação de computadores independentes para resolverem um problema em comum. Os *clusters* podem ser utilizados para sistemas *High Availability(HA)* para garantir alta disponibili-

dade do sistema ou em *High Performance Computing (HPC)* para proporcionarem poder computacional superior do que um único computador proporcionaria (STERLING, 2002).

Os *clusters beowulf* são de desempenho escalável baseados em *hardware* facilmente encontrado no mercado, em sistemas de redes comuns e tendo como infraestrutura o *software* livre. Os computadores que formam um *cluster* são mais fracamente acoplados do que as unidades de um sistema *NUMA*, desta forma, permitindo maior escalabilidade. Os *clusters beowulf* possuem alta adaptabilidade, podendo ser formados por dois nodos conectados via *ethernet* ou ser um complexo sistema de 1024 nodos conectados através de rede de alta velocidade (BEOWULF... , 2004).

A comunicação entre os nodos de um *cluster beowulf* é feita através de bibliotecas de troca de mensagens. Atualmente o principal padrão é o *Message Passing Interface (MPI)* (MESSAGE... , 2006). Possui diversas implementações que são utilizadas como bibliotecas nos programas a serem implementados, fazendo abstração da comunicação entre os nodos. É importante ressaltar que os softwares executados em *clusters beowulf* devem ser preparados para isto, utilizando algoritmos para processamento distribuído e tendo na sua implementação, uma biblioteca para a comunicação entre os nodos.

O *Blast* possui uma versão que utiliza o padrão *MPI*. Esta versão, chamamada *mpi-Blast* (DARLING; CAREY; FENG, 2003), divide o banco de sequências em partes iguais e no momento de execução, cada conjunto de partes é alocado a um processo que é executado no *cluster*. Outro *software* que possui uma versão para *clusters* é o programa de alinhamentos múltiplos *ClustalW* (CHENNA et al., 2003) em sua versão *ClustalW-MPI* (LI, 2003). Para inferência filogenética por *maximum likelihood* há trabalhos como o de Stamatakis (2004) e Du, Lin e Roshan (2005) e para matrizes de distância o trabalho de Albrecht, Hübner e Dávila (2007).

3.1. Grids

Grids são agrupamentos computacionais mais fracamente acoplados que os *clusters* e estão sendo utilizados na computação científica. Eles são úteis quando há uma grande quantidade de dados a serem processados e que não haja urgência para o recebimento dos resultados, ou seja, para uma pequena quantidade de dados eles não são vantajosos por causa do tempo de resposta, porém, quando há um grande volume de dados, ocorre a divisão destes dados em vários ambientes computacional e o resultado é retornado mais rapidamente do que se fosse executado num ambiente computacional próprio.

Existem esforços de construção de grandes *grids* computacionais, por exemplo o *EGEE*, onde o foco é o uso de simulações de engenharia e ciências, inclusive bioinformática. Para utilizar *grids*, normalmente define-se *workflows* onde descreve-se o fluxo dos dados e os programas a serem executados através de um diagrama. Desta forma, as fases da execução são distribuídas dentro de um *grid* e são executadas de forma *MIMD*, ou seja, diversas execuções dos *softwares* sendo executadas nos sistemas computacionais do *grid*, cada qual com uma fatia dos dados, havendo assim um paralelismo em nível de dados para o mesmo *software* e podendo haver um paralelismo em nível de programa, pois quando há uma quantidade de dados suficientes, fazes seguintes do *workflow* já podem ser executadas.

4. Uso de processadores gráficos

Os *GPU* são otimizados para processamento gráfico e sua arquitetura é diferente dos processadores de propósito geral, porém, seu uso em aplicações não gráficas está em fase de crescimento. Isto ocorre por alguns motivos: o grande aumento do desempenho destes processadores, a inclusão destes processadores no maior número de computadores, o preço competitivo em relação aos processadores de propósito geral e as novas possibilidades de programação destes dispositivos.

Segundo GPGPU (2007), com o aumento da comodidade de programação das unidades de processamento gráfico (*GPU*), estes chips estão capazes de executar mais do que computações gráficas específicas para qual foram desenvolvidos. Eles são agora úteis como co-processadores e suas altas velocidades fazem eles úteis a uma variedade de aplicações.

Na bioinformática, o uso de processadores gráficos como co-processadores é útil para a resolução de sistemas numéricos (GALOPPO et al.,) com matrizes e existe trabalhos que os utilizam em aplicações de bioinformática, por exemplo Charalambous, Trancoso e Stamatakis (2005).

5. Utilização de múltiplos níveis

Desenvolvedores e pesquisadores que utilizam tais tecnologias muitas vezes fazem uso de apenas um nível de paralelismo nos *clusters* ou *grids*, ignorando outras possíveis otimizações. Para obter os benefícios dos processadores, não são necessárias grandes mudanças nos *softwares*, basta algumas pequenas mudanças.

Muitos recursos dos processadores são desperdiçados quando compila-se e constrói-se o *softwares* com o objetivo de obter-se o maior número de processadores compatíveis. Os compiladores e montadores atuais, constroem os *softwares* para que eles tenham compatibilidade com os antigos processadores 386. Desta forma, otimizações, como *pipelines* e conjunto de instruções *SIMD* são sub ou não utilizados. Desta forma, recomenda-se compilar o aplicativo tendo em mente as máquinas alvos, gerando diversos binários diferentes, um para cada modelo de processador. Outra solução é manter os fontes abertos, como grande parte dos *softwares* de bioinformática são livres, e ter instruções de como compilar para fazer uso das otimizações da plataforma alvo. É importante lembrar de usar variáveis que não sejam fortemente acopladas e nem muitos desvios, para não prejudicar o uso de *pipelines*.

Para utilizar os processadores *multicores*, recomenda-se que o programa primeiramente não faça uso de variáveis globais e que ele seja construído através de pequenas funções específicas. Então, separar as funções de envio e recebimento de dados dos padrões *MPI* ou de paralelismo em funções específicas. E para melhor utilizar as *grids* computacionais, é melhor o uso de programas específicos, ou seja, cada passo da execução deve executar uma única tarefa, desta forma fica mais fácil escalar os dados em múltiplos programas e o desenvolvimento de otimizações específicas, por exemplo, o uso de instruções *SIMD* e do processador gráfico.

É importante ressaltar que o uso dos paralelismo não são mutualmente exclusivos, ou seja, um *workflow* pode utilizar paralelismo na execução de múltiplas instâncias de um *software*, cada qual com um conjunto de dados diferentes. Cada um dos *softwares* deve

utilizar das tecnologias *MPI* para comunicação entre diversos computadores que integram o *cluster* e cada um destes computadores serem compostos de mais de um processador *multicore*. E os *softwares* devem fazer uso das otimizações das instruções do processador.

6. Olhando a frente

Com o fim da disputa por *Mhz*, os desenvolvedores de processadores e outros componentes computacionais estão investindo maciçamente no aumento de núcleos de processamento dos processadores, sendo que começou com dois núcleos, já é possível adquirir processadores com quatro e está previsto para o terceiro trimestre de 2007, processadores com 8 núcleos para usuários domésticos. Desta forma, os aplicativos de bioinformática devem ser repensados para fazer uso de tais mudanças e melhorias.

Um dos motivos da mudança de foco dos desenvolvedores, na diminuição investimento em aumentos de *Mhz*, é que o aumento da quantidade de ciclos por segundo nos processadores resulta num aumento do consumo de energia por este componente. Uma mudança de paradigma que está tomando força, é a venda de processadores e unidades computacionais não somente pelo seu desempenho total, mas pela relação de *Watts* consumidos por *Million instructions per second (MIPS)*, ou seja, a relação de consumo de energia pela quantidade de computações executadas. Esta é uma mudança que tende a crescer, pois cada vez o custo de energia está maior e as empresas necessitam de maior poder computacional.

Outras mudanças são as otimizações nos conjunto de instruções da arquitetura *IA – 32*, onde está previsto uma extensão para o *SSE*(INTEL, 2007a) em que uma das melhorias, são instruções para pesquisa em textos e por padrões, tarefas comuns na bioinformática. Porém, para fazer uso destas otimizações, os programas desenvolvidos devem estar preparados através da utilização de instruções diretas ao processador ou com o uso das otimizações do compilador.

7. Comentários finais

Neste trabalho foram apresentados os principais níveis de paralelismo disponíveis nos processadores e ambientes computacionais de pesquisa de bioinformática. Utilizar tais otimizações e paralelismo nos *softwares* de bioinformática e seus *workflows* não é algo trivial, porém com as novas ferramentas, padrões e bibliotecas para utilização de paralelismo e até mesmo de processadores gráficos, espera-se utilizar melhor os recursos disponíveis nos microcomputadores atuais. Desta forma, deve-se conseguir acompanhar o crescimento das informações bio-moleculares disponíveis e assim alcançar novos patamares na bioinformática.

Referências Bibliográficas

ALBRECHT, F. F.; HÜBNER, J. F.; DÁVILA, A. M. R. A distributed algorithm for phylogenetics inference. In: BRAZILIAN SYMPOSIUM ON BIOINFORMATICS, 2007, Angra dos Reis, RJ. *BSB 2007 Poster Proceedings*. Angra dos Reis, RJ: Brazilian Computer Society, 2007. p. 66–69.

ALTSCHUL, S. F. et al. Gapped blast and psi-blast: a new generation os protein database search programs. *Nucleic Acids Res.*, v. 25, p. 3389–3402, 1997.

- ANDREWS, G. R. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Arizona: Addison-Wesley, 2000. 664 p.
- BEOWULF.ORG: the Beowulf cluster site. San Francisco: Beowulf.org, 2004. Disponível em: <www.beowulf.org>. Acesso em: 26 mar. 2006.
- CHARALAMBOUS maria; TRANCOSO, P.; STAMTAKIS, A. Initial experiences with porting a bioinformatics application to a graphics processor. [Http://www.gpgpu.org](http://www.gpgpu.org). 2005.
- CHENNA, R. et al. Multiple sequence alignment with the clustal series of programs. *Nucleic Acids Res*, Illkirch, v. 31, n. 13, p. 3497–3500, mar. 2003.
- DARLING, A.; CAREY, L.; FENG, W. The desing, implementation, and evaluation of mpiblast. In: INTERNATIONAL CONFERENCE ON LINUX CLUSTERS: THE HPC REVOLUTION 2003 IN CONJUNCTION WITH THE CLUSTERWORLD CONFERENCE & EXPO, 4., 2003, San Jose, California, USA. *Proceddings...* San Jose, CA: LA-UR, 2003. p. 20–34.
- DU, Z.; LIN, F.; ROSHAN, U. W. Reconstruction of large phylogenetic tree: A parallel approach. *Computational Biology and Chemistry*, Singapore, v. 29, p. 273–280, Jan. 2005.
- FOUNDATION, F. S. *GCC, the GNU compiler collection*. Boston: [s.n.], 2006. Disponível em: <<http://gcc.gnu.org/>>. Acesso em: 31 out 2006.
- FOUNDATION, F. S. *GCC-Gomp*. 2007. Disponível em: <<http://gcc.gnu.org/projects/gomp>>. Acesso em: 01 set 2007.
- GALOPPO, N. et al. Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware. [Http://www.gpgpu.org](http://www.gpgpu.org).
- GPGPU. 2007. Disponível em: <<http://www.gpgpu.org/>>. Acesso em: 31 ago. 2007.
- HEBENSTREIT, M. Programas orquestrados. *Linux Magazine*, São Paulo, n. 33, p. 44–49, ago. 2007.
- HENNESSY, J.; PATTERSON, D. *Organização e Projeto de Computadores: A interface hardware/software*. 2. ed. Rio de Janeiro: Editora LTC, 1998. 551 p.
- HENNESSY, J.; PATTERSON, D. *Arquitetura de Computadores: Uma abordagem quantitativa*. 3. ed. Rio de Janeiro: Editora Campus, 2003. 827 p.
- INTEL. *IA-32 Intel® Architecture Software Developer's Manual: Basic architecture*. [S.l.], June 2005. v. 1. Order Number: 253665-016.
- INTEL. *Intel® SSE4 Programming Reference*. [S.l.], July 2007. v. 1. Reference Number: D91561-003.
- INTEL. *Moore's Law, the future*. Denver, CO: [s.n.], 2007. Disponível em: <<http://developer.intel.com/technology/silicon/>>. Acesso em: 31 ago 2007.
- LI, K.-B. Clustalw-mpi: clustalw analysis using distributed and parallel computing. *Bioinformatics Application Note*, [S.I.], v. 19, n. 12, p. 1585–1586, mar. 2003.
- MESSAGE Passing Interface. Chicago: [s.n.], 2006. Disponível em: <www-unix.mcs.anl.gov/mpi/>. Acesso em: 26 mar. 2006.

OPENMP. *OpenMP*. 2007. Disponível em: <<http://www.openmp.org>>. Acesso em: 01 set 2007.

RAMANATHAN, R. *Extending the World's Most Popular Processor Architecture: New innovations that improve the performance and energy efficiency of intel® architecture*. [S.l.], 2007.

REINDERS, J. *Intel Threading Building Blocks: Outfitting c++ for multi-core processor parallelism*. Sebastopol, CA: O'Reilly Media, Inc., 2007. 332 p.

ROGNES, T.; SEEBERG, E. Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, v. 16, n. 8, p. 699–706, Mar. 2000.

STAMATAKIS, A. *Distributed and parallel algorithms and systems for inference of huge phylogenetic trees bases on the maximum likelihood method*. 132 p. Tese (Ph.D. on Computer Science) — Technischen Universität München, München, 2004.

STERLING, T. (Ed.). *Beowulf cluster computing with Linux*. Cambridge, Massachusetts: The Mit Press, 2002. 496 p.