Mapeamento de vetores em malhas bidimensionais *

Andréa Iabrudi Tavares Fabiano Cruz Peixoto
Márcio Luiz Bunte de Carvalho Osvaldo Sérgio Farhat de Carvalho

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte - MG
iabrudi@dcc.ufmg.br

Resumo

Nesse artigo apresentamos um mapeamento de vetores numa máquina SIMD em malha bidimensional, a Zephyr. O mapeamento proposto permite que algoritmos projetados para uma máquina SIMD vetorial sejam executados eficientemente na Zephyr, explorando seus recursos de modo racional. As alterações necessárias para implementação do mapeamento são simples e estão descritas no texto. Os resultados obtidos para o algoritmo de cálculo de similaridade de sequências de DNA mostram que nossa abordagem tem a execução bem mais eficiente do que a conseguida quando o mapeamento automático da máquina é utilizado.

Abstract

In this article we describe the mapping of vectors into a SIMD mesh-connected 2D machine, the Zephyr. Our approach explore the machine's resources efficiently, allowing that algorithms designed for a SIMD vetorial run with a good performance in Zephyr. The changes in the program are simple and are completly estabilished in the text. The results achieved with the similarity of DNA sequences algorithm are significantly better when our mapping is used.

*Esse projeto foi parcialmente financiado pelos órgãos FAPEMIG, CNPq e CAPES

1 Introdução

No projeto de algoritmos paralelos é imprescindível que se leve em conta a arquitetura da máquina alvo. O efeito da comunicação entre processadores, assim como o tipo de sincronização que existe entre eles é determinante para o bom desempenho do algoritmo. Contudo, ao se considerar um novo modelo de arquitetura, muitas vezes é possível adaptar soluções sequenciais, ou mesmo propostas paralelas desenvolvidas para outra máquina, conseguindo-se um ganho significativo de desempenho, sem um esforço grande de projeto. Mesmo correndo o risco de não conseguir um algoritmo de melhor complexidade possível para a arquitetura específica, essa solução adaptativa permite que melhoras sejam obtidas rapidamente, enquanto se procura pelo algoritmo ótimo, que muitas vezes não é óbvio.

Se é mais fácil projetar um algoritmo quando usamos como estrutura de dados um vetor, havendo dependência somente entre posições contíguas e aplicando-se as mesmas operações sobre todos os elementos, a melhor máquina para resolver o problema seria uma SIMD em malha unidimensional, ou vetorial. Uma maneira rápida de conseguir um algoritmo equivalente para outra arquitetura SIMD, obtendo-se um bom desempenho, é mapear o vetor para os processadores da máquina, tentando colocar elementos de posições vizinhas em processadores diretamente conectados. Estamos diante do problema de mapear uma estrutura unidimensional em outra estrutura topólogica, tentando manter as características de uma SIMD em malha unidimensional: comunicação entre elementos consecutivos do vetor são sempre feitas com uma instrução e entre processadores conectados fisicamente.

A possível perda em desempenho da abordagem proposta acima pode ser compensada pela diminuição do tempo de projeto de um algoritmo apropriado para a nova arquitetura. Com o mapeamento do vetor, pode-se aproveitar algoritmos já existentes tentando obter o melhor desempenho possível da máquina, com pequeno esforço de programação. Nesse artigo abordamos o problema de mapear-se um vetor em uma arquitetura SIMD em malha bidimensional, tentando diminuir as diferenças existentes entre as duas arquiteturas. Para isso, apresentamos um assinalamento que determina uma distância um entre processadores que contenham elementos vizinhos, onde cada passo de comunicação pode ser feito sempre com três instruções. Os algoritmos já existentes para uma SIMD vetorial podem ser traduzidos rapidamente, sendo necessárias alterações na parte de inicialização dos dados e instruções de comunicação.

A máquina utilizada é a Zephyr da Wavetracer[5], composta de uma malha bidimensional de 8K processadores. As estruturas de dados manipuladas são matrizes de quaisquer dimensões, sendo criado um processo para gerenciar cada uma das posições. Se o número de processos excede o número de processadores da malha, um mapeamento é feito de modo transparente, aproveitando a estrutura de interconexão da máquina. O espaço mínimo de alocação corresponde à configuração física dos processadores, ou seja, uma matriz de 128×64. É possível a declaração de vetores, mas esses são alocados numa matriz onde somente uma linha está efetivamente ocupada com dados. Pode-se então desenvolver algoritmos como se a máquina fosse uma SIMD vetorial, mas, como veremos a seguir, essa abordagem subutiliza a maioria dos processadores da malha física, levando a um desempenho medíocre. Apresentamos uma proposta que utiliza todos os processadores, diminuindo o impacto dos passos de comunicação. Mostramos a melhora significativa de desempenho quando o algoritmo implementado trabalha sobre vetores.

São apresentados testes comparativos entre o mapeamento fornecido pela máquina e nossa proposta, na solução de dois problemas. O primeiro deles é o algoritmo de ordenação par-ímpar [2], caso em que a melhora do tempo de execução em paralelo foi significativa. Contudo, existe uma proposta para ordenação em SIMD bidimensional com complexidade $O(\sqrt{n})$ [4]. O outro é o algoritmo para determinação da similaridade entre duas sequências de DNA, utilizando técnicas de programação dinâmica [3]. Esse problema é bastante importante no campo da biologia, servindo como base para determinação de homologia entre novas cadeias de DNA sequenciadas, para elaboração de árvores evolucionárias ou para procura de certos tipos de padrões que determinam uma funcionalidade específica da proteína sintetizada pela cadeia. Não se tem conhecimento de uma proposta de implementação paralela para o algoritmo, mas suas características permitem uma tradução quase direta da implementação sequencial, quando a arquitetura alvo é uma SIMD

vetorial. Assim, o mapeamento proposto facilitou o projeto, permitindo-se explorar de modo racional os recursos da máquina, sem aumentar muito a complexidade de programação. Os tempos obtidos foram melhores do que os da implementação sequencial, com resultados satisfatórios, principalmente quando o tamanho da entrada do problema cresce.

Na próxima seção apresentamos uma descrição da máquina Zephyr, abordando aspectos de hardware e software. A proposta de mapeamento de vetores na máquina é apresentada na seção seguinte, além de uma descrição do impacto no estilo de programação. São mostrados então os resultados obtidos para os dois problemas propostos. Conclusões e propostas para trabalhos futuros finalizam o artigo.

2 A Máquina Zephyr

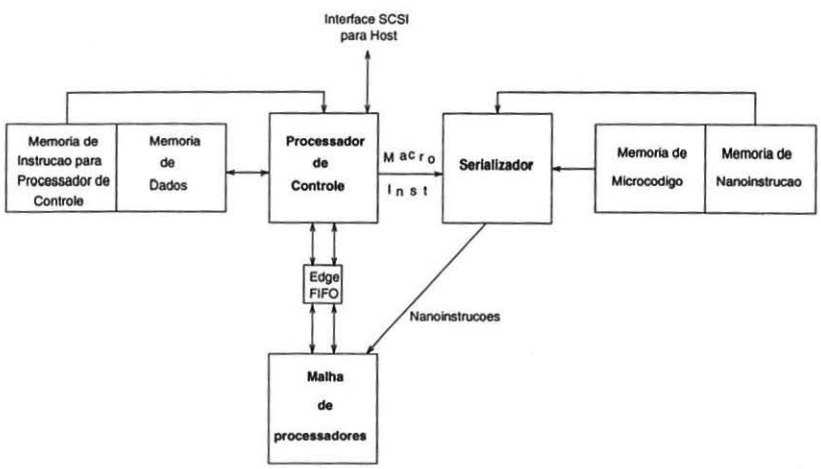


Figura 1: Componentes da Máquina Zephyr

A máquina Zephyr é uma máquina SIMD em malha, cujos elementos de processamento são fracamente acoplados. A malha pode tomar configurações bi ou tridimensionais, controladas por software, de acordo com especificação do programador. Nesse artigo estaremos interessados na sua configuração bidimensional, concentrando nossas discussões nela. Como mostrado na figura 1, a máquina é composta de quatro módulos principais[1]:

1. **Malha de processadores** Cada elemento de processamento é um processador de 1 bit com 2Kbits de memória local e 203Kbits de memória estendida. Na configuração bidimensional, consiste de uma malha de dimensões 128×64.

2. **Processador de Controle** Responsável pela comunicação com a estação de trabalho *host*, através de interface SCSI, e com a malha, através de Edge FIFO. Controla a execução das macroinstruções e gerencia os dados compartilhados pelos processadores.

3. **Serializador** Responsável pela tradução das microinstruções em nanoinstruções e pela distribuição dessas para a malha de processadores. Há uma implementação de *pipeline* que aumenta ainda mais o paralelismo da máquina.

4. **Edge FIFO** Interface bidirecional de comunicação entre o processador de controle e a malha de processadores, permite a transmissão simultânea de dados para 64 processadores (coluna da matriz), aumentando a rapidez na comunicação de dados.

Ao se executar um programa, são definidas as dimensões do espaço de processamento sobre o qual a malha de processadores irá trabalhar. Pensando-se nesse espaço como uma matriz, a cada posição é associado um processo, que fica responsável pela manipulação dos dados atribuídos a ela. Um processo é composto pelo código local de execução e pelos dados da posição da matriz. O **processador de controle** é responsável pelo assinalamento de processos a processadores da malha. Cada processador usa 410 bits de sua memória local para código de gerenciamento, mais 3 bits para cada processo que lhe é atribuído, necessários para rotinas de escalonamento. O restante da memória (local e estendida) é repartida entre os processos, cabendo a cada um bloco de mesmo tamanho.

Se as dimensões do espaço de processamento são menores ou iguais às dimensões da malha, somente um processo é associado a cada processador, num mapeamento direto entre as posições do espaço e a localização do processador na malha. O **processador de controle** é responsável pelo armazenamento e manipulação de dados globais, pelo controle de comunicação e ativação dos processos e pela decodificação das macroinstruções. Fica a cargo do serializador o gerenciamento do *pipeline*, sendo ele que transmite as nanoinstruções para os processadores da malha.

Se qualquer das dimensões do espaço de processamento exceder a dimensão correspondente na malha, é necessário um outro mecanismo de mapeamento, associando mais de um processo a cada processador. A memória física disponível é divida em blocos de mesmo tamanho, de acordo com o número de processos. O processador, mudando de um bloco para outro, altera seu contexto, executando um processo de cada vez. Dadas as características de sincronização de uma máquina SIMD, cada processador tem que gerenciar o mesmo número de processos. Era de se esperar que o tempo de execução aumentasse linearmente com o número de processos por processador.

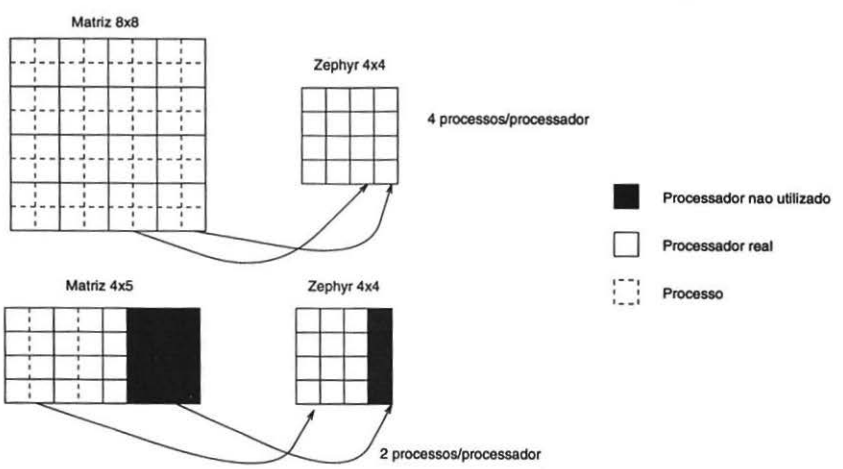


Figura 2: Mapeamentos automáticos de matrizes

Contudo, o mecanismo de assinalamento de processos é feito de modo a diminuir a necessidade de comunicação interprocessadores, colocando sempre que possível, processos referentes a posições vizinhas no espaço num mesmo processador. Isso é obtido dividindo-se esse espaço em submatrizes e atribuindo a cada processador uma delas (figura 2). Note-se que o máximo de eficiência é obtido quando as dimensões do espaço de processamento são múltiplos das dimensões da malha, caso em que nenhum processo ou processador fica inativo. Com o mapeamento através de submatrizes, muitas das comunicações entre processos são feitas dentro do mesmo processador, de maneira mais eficiente, sendo que o aumento no tempo de execução normalmente é sublinear em relação ao número de processadores utilizados.

As ferramentas fornecidas para a programação da Zephyr são duas: um compilador, multiC [6], e um software de monitoração, o dtcmonitor. Esse último permite verificar o número de bits que um processo está utilizando, quantos processos estão associados a cada processador e as dimensões do espaço de processamento. O multiC é um super-conjunto do C padrão. A linguagem permite a utilização de todas as facilidades de programação do C, além de fornecer extensões para programação SIMD. Fica a cargo programador estabelecer as dimensões do espaço de processamento e o algoritmo em alto nível, baseado nas seguintes extensões fornecidas:

1. O especificador `multi` que declara um valor independente de variável para cada processo. Por exemplo, a declaração `multi int a` associa uma variável inteira `a` para cada processo.

2. O operador de comunicação entre processos, definindo a distância em cada uma das dimensões (número de passos de comunicação). O formato do operador é `[dx, dy]`, onde `dx` define o deslocamento na linha e `dy`, na coluna. O comando `a = [1, -1]b` atribui à variável `a` de cada processo, o valor da variável `b` do processo uma coluna à direita(`dx`) e uma linha acima(`dy`).

3. Operadores de redução, que retornam um valor `uni`, variável única global a todos os processos, depois de uma operação sobre todos os valores de uma variável `multi`.

4. Funções para determinação da localização lógica do processo no espaço de processamento, que é visto como uma matriz do C.

5. Função de inicialização do processamento em paralelo, `multi_perform`, que determina as dimensões do espaço de processamento e executa uma rotina na máquina Zephyr.

Os valores `uni`, assim como maior parte do código da função multiprogramada (realizada com `multi_perform`) são guardadas na memória do **processador de controle**. Já as variáveis `multi` são guardadas na memória dos processos. O programador não tem nenhum controle sobre o método de mapeamento, que fica completamente a cargo do processador de controle.

3 O Mapeamento Serpentina

Como visto na seção anterior, a máquina Zephyr provê recursos de mapeamento para estruturas bidimensionais. É possível considerá-la como uma SIMD vetorial, bastando para isso declarar-se um espaço de processamento com uma das dimensões de tamanho 1. Contudo, o mapeamento executado pela máquina deixa inativos muitos processadores, enquanto outros ficam sobrecarregados (figura 3). Isto ocorre porque a dimensão mínima alocada pela máquina é uma matriz 128×64, e uma estrutura unidimensional será sempre alocada na primeira linha de processadores. Neste caso, ao definirmos um vetor de 129 posições, 8054 processadores estarão inativos, enquanto que os 128 ativos controlarão dois processos cada (figura 3).

O mapeamento fornecido a seguir permitirá que a programação seja feita tendo em vista uma arquitetura SIMD vetorial, mas fornecendo meios para que os processadores não fiquem inativos. Faremos uma exploração racional da arquitetura disponível, sem dificultar o trabalho de programação e utilizando os recursos de mapeamento bidimensional já fornecidos pela máquina. A abordagem mais ingênua de se declarar um vetor, além de subutilizar os processadores da malha, degrada ainda mais o tempo de execução devido à necessidade de se utilizar a memória estendida antes que a soma das memórias locais seja esgotada. Apesar de não anunciado pelo fabricante, a memória estendida implica em uma modificação grande da constante de multiplicação da complexidade do tempo de execução, além do fator de degradação estabelecido para aumento do número de processos controlados por cada processador.

Para permitir que um algoritmo desenvolvido para uma arquitetura SIMD vetorial seja executado com eficiência em uma outra arquitetura SIMD, devemos conseguir um mecanismo para que as comunicações na primeira sejam executadas de modo eficiente na última. Esse é o ponto mais importante, pois as outras características de sincronização e de execução de uma única instrução

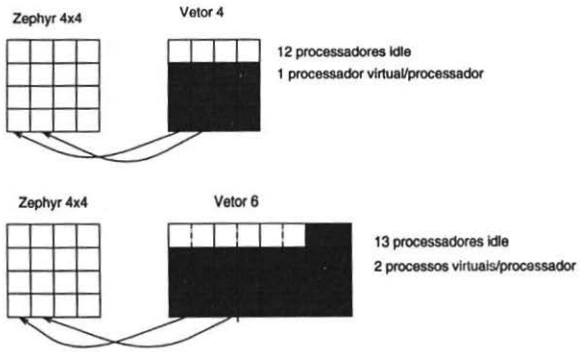


Figura 3: Mapeamentos automáticos de vetores

para dados diferentes são conservadas. Numa SIMD vetorial, a estrutura de conexão é linear, ou seja, cada processador é conectado com seu antecessor e seu sucessor. Se fazemos uma associação entre elementos do vetor e processadores, temos que cada elemento deve ter comunicação direta com os elementos das duas posições vizinhas.

No estabelecimento do mapeamento entre posições do vetor com posições do espaço de processamento, levando-se em conta as características de sincronização e que executa-se a mesma instrução para todos os processos numa máquina SIMD, dois pontos são relevantes:

1. Minimização dos passos de comunicação entre processos responsáveis por posições vizinhas do vetor. Idealmente, o número de passos deve ser 1.

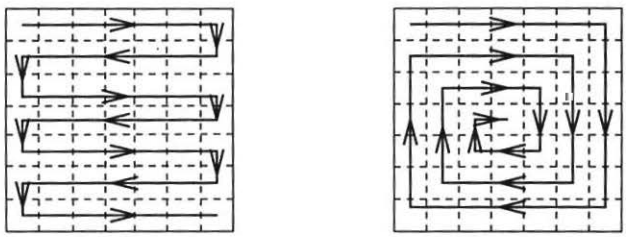
2. Regularidade do mapeamento. Para um determinado sentido de comunicação entre elementos vizinhos (antecessor ou sucessor), as opções de direção de comunicação real devem ser minimizadas, pois cada direção implicará em uma instrução SIMD distinta.

Com base nas observações acima, a solução proposta para o mapeamento foi a de uma "serpentina", como mostrado na figura 4.a. Os elementos do vetor são distruibuídos nas linhas da matriz, alterando-se o sentido da distribuição (para direita ou para esquerda) a cada linha. Desse modo, os vizinhos de um processador são sempre acessíveis em um passo de comunicação, como no caso da estrutura física vetorial. Para o acesso paralelo aos vizinhos numa determinada direção, existirão sempre três direções de acesso real: direita e esquerda, dependendo da linha em que o processador se encontra na matriz, e para cima (predecessor) ou para baixo (antecessor), no caso dos processadores das bordas da matriz. Note-se, por exemplo, o mapeamento "caracol" mostrado na figura 4.b, que mantém distância um entre os processadores, mas determina quatro diferentes direções a cada comunicação.

A comunicação de posições vizinhas no vetor é realizada em um passo, executado em três etapas (instruções distintas). A perda em tempo de execução acarretada pelo aumento dos números de etapas é compensada pela possibilidade de se utilizar os outros processadores da malha, antes inativos. A dimensão máxima obtida, sem necessidade de mais de um processo por processador é de 8192 em oposição ao 128 inicial, mas é possível aproveitar os mecanimos de mapeamento automático da máquina para declaração de vetores de dimensões maiores.

A tradução de algoritmos desenvolvidos para máquinas SIMD vetoriais é bastante simples, se concentrando em dois pontos. Primeiramente, a declaração das dimensões do espaço de processamento e a atribuição inicial das posições do vetor a posições do espaço, a fim de refletir a distribuição do mapeamento serpentina.

Depois, deve-se traduzir os operadores de comunicação, para que reflitam a disposição relativa dos elementos do vetor no espaço de processamento. Isso pode ser feito facilmente, observando-se



a) Serpentina **b)Caracol**

Figura 4: Dois mapeamentos com distância um

a seguinte tradução, considerando x como a coordenada de coluna no espaço de processamento declarado:

```
/* elemento sucessor */
se linha par e x != limite direito
  [+1]a muda para [+1,0]a
se linha impar e x != limite esquerdo
  [+1]a muda para [-1,0]a
senao
  [+1]a muda para [0,+1]a

/* elemento predecessor */
 se linha par e x != limite direito
   [-1]a muda para [-1,0]a
 se linha impar e x != limite esquerdo
   [-1]a muda para [+1,0]a
 senao
   [-1]a muda para [0,-1]a
```

4 Resultados

Os resultados desta seção apresentam os tempos medidos para implementações em multiC na máquina Zephyr. Os tempos apresentados para as computações paralelas são tempo de resposta real, já que não há uma função própria para tomada de tempo de CPU para a Zephyr. Para isso, a máquina *host* foi isolada para que cuidasse apenas dos processos executados na Zephyr. No caso da comparação com algoritmo sequencial, a máquina utilizada foi uma Sun SparcStation II, com compilador `cc` e opção de otimização `-O4`. Nesse caso, foram medidos os tempos de CPU, através do comando `time` do UNIX.

Nos gráficos apresentados, os testes foram feitos para tamanho da entrada variando de passos de 128, já que a dimensão sendo um múltiplo do tamanho real da malha, o máximo de desempenho é obtido (veja seção de mapeamento). No caso do mapeamento automático, somente uma linha da malha de processadores (128) foi utilizada, como discutido anteriormente. O mapeamento serpentina utilizou-se de toda a malha de processadores (8192). A única diferença entre as duas implementações se refere à inicialização dos valores da matriz e à tradução dos operadores de comunicação.

4.1 Ordenação Par-Ímpar

A implementação sequencial do algoritmo de ordenação par-ímpar para n elementos é $O(n^2)$[2]. Sua implementação em uma SIMD vetorial é $O(n)$, utilizando n processadores, já tendo sido

proposta na literatura [4]. O gráfico da figura 5 mostra a diferença de tempo de execução do algoritmo de ordenação par-ímpar implementado com o mapeamento automático da máquina e com o mapeamento serpentina.

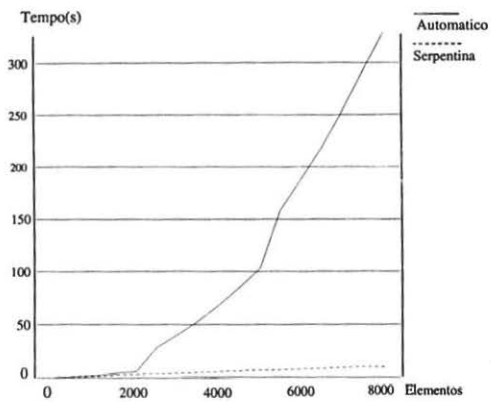


Figura 5: Gráfico de Ordenação: Número de Elementos × Tempo de Execução

Como se pode notar, as duas implementações têm um comportamento bastante semelhante enquanto os vetores possuem menos de 2048 elementos, sendo que o mapeamento fornecido pela Zephyr é um pouco inferior. Isso se deve à utilização de muitos processos por processador, pois a cada intervalo de 128 elementos, mais um processo é criado. Note-se a implementação serpentina, mesmo se utilizando de três etapas a cada comunicação, tem um comportamento superior.

A partir do ponto 2048 o tempo de execução do mapeamento padrão é bastante pior. Esse ponto corresponde ao início da utilização da memória estendida. A complexidade do algoritmo não se altera, contudo a constante de multiplicação é bem maior do que na memória local. Além disso, parece que essa constante se altera durante o decorrer do processamento. Esse fato não pode ainda ser explicado com base na documentação existente sobre a máquina. Já o mapeamento serpentina fornece um comportamento bastante estável. Durante todos os testes, somente um processo por processador foi utilizado, e podemos notar a grande diferença entre os tempos medidos. Como esperado, o *speedup* obtido não é diretamente proporcional ao aumento do número de processadores, já que cada passo de comunicação leva, a princípio, três vezes mais tempo para executar.

4.2 Similaridade entre Sequências de DNA

O cálculo de similaridade entre duas cadeias de DNA é um problema bastante importante na biologia. A implementação mais eficiente é a que utiliza programação dinâmica e o algoritmo sequencial foi implementado segundo proposto em [3]. O algoritmo calcula uma matriz de similaridades relativas as diversas possibilidades de casamento entre as duas sequências, quando se permitem inserções, deleções ou substituições de bases. Ao final do processamento, na última posição da matriz está expresso o grau de similaridade entre as duas sequências. As operações para determinar o valor de uma posição são sempre as mesmas, alterando-se apenas o valor dos operandos utilizados. Sejam as sequências A de m bases e B de n bases. Para o cálculo da similaridade, utiliza-se uma matriz $m \times n$, onde o valor da posição $[i, j]$ depende dos valores das posições $[i, j-1]$, $[i-1, j]$ e $[i-1, j-1]$. A implementação sequencial é $O(n \times m)$, utilizando espaço $O(n)$. Não se tem conhecimento de implementações paralelas para o algoritmo, e devido a sua importância e larga utilização seria interessante conseguir uma solução mais eficiente.

A implementação SIMD em malha bidimensional é direta. Associa-se um processo a cada posição da matriz de similaridade e, como as instruções são as mesmas para todas as posições, basta definir o número de passos de execução determinado pela dependência entre os dados. Seja a matriz $m \times n$ e a diagonal D_i definida pelos elementos cuja soma dos índices na matriz é igual a i. A dependência dos dados pode ser expressa em termos dessas diagonais, ou seja, os elementos de D_i dependem dos elementos de D_{i-1} e D_{i-2}. Como existem $m+n-1$ diagonais distintas, o limite inferior de complexidade para o algoritmo paralelo é $O(m+n)$. Nessa abordagem, estaríamos utilizando $m \times n$ processadores.

Porém o número máximo de elementos em uma diagonal é m (número de linhas da matriz). Assim, é possível uma implementação que utilize apenas m processadores, onde a diagonal tratada a cada passo é guardada num vetor. Chegamos então a uma implementação que continua com complexidade $O(m+n)$ utilizando espaço $O(m)$. O algoritmo paralelo para solucionar o problema de similaridade foi implementado tendo em vista uma arquitetura SIMD vetorial, dada a facilidade de projeto. Depois utilizou-se os conceitos de mapeamento serpentina para mapeá-lo para uma estrutura bidimensional.

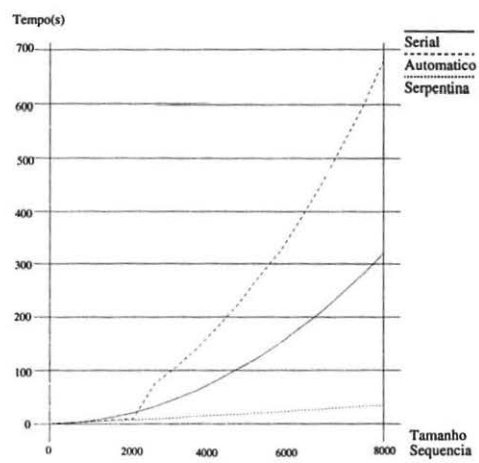


Figura 6: Gráfico do Algoritmo de Alinhamento de Sequências

No gráfico 6 podemos examinar as curvas do tempo de execução dos algoritmos, quando comparando duas sequências de tamanho n. Como esperado, o sequencial é $O(n^2)$, com comportamento bastante estável. A implementação paralela utilizando o mapeamento automático mostra-se rapidamente inferior ao desempenho sequencial, fato explicado pela utilização de memória estendida. Já a implementação com mapeamento serpentina supera a implementação sequencial e cresce linearmente, como esperado.

5 Conclusão

A máquina Zephyr nos fornece um bom exemplo de que o conhecimento da estrutura de hardware e da operação do software de uma máquina paralela é imprescindível para elaboração de algoritmos paralelos eficientes. A possibilidade de utilização de espaço de processamento com dimensão igual a 1, ou seja, vetorial, pode levar a falsa impressão que a máquina procura otimizar o mapeamento de tal espaço. Muitas das experiências realizadas anteriormente com a Zephyr, por falta de documentação apropriada e de um estudo mais detalhado do mecanismo automático de mapeamento,

causaram frustações com os resultados obtidos, pois o uso de memória estendida abaixa bastante o desempenho da máquina.

Nesse artigo apresentamos uma proposta de mapeamento de vetores no espaço de processamento da Zephyr, que permite a tradução de algoritmos elaborados para SIMD vetorial de maneira rápida e explorando eficientemente as características da máquina. A necessidade de se preocupar com a arquitetura para a qual o algoritmo está sendo desenvolvido é totalmente reconhecida, e sabemos que na maior parte das vezes nossa abordagem levará a resultados que poderiam ser melhores, caso a concepção do algoritmo fosse alterada no sentido de se usar realmente o espaço bidimensional da máquina. Contudo, como mostrado na seção anterior, se um algoritmo novo é facilmente projetado para SIMD vetorial e é de concepção mais elaborada para Zephyr, o mapeamento serpentina permite explorar de modo racional os recursos oferecidos, causando uma grande melhora de desempenho. No caso específico do cálculo de similaridade de sequências de DNA, a abordagem mais ingênua de utilizar o mapeamento automático fornecido pela máquina mostrou-se bastante inferior ao desempenho do algoritmo sequencial, enquanto que o mapeamento serpentina apresentou resultados superiores.

Pretendemos estender o trabalho até aqui realizado fornecendo um pré-processador de programas projetados para uma versão virtual da Zephyr, com configuração unidimensional de 8192×1. Esse pré-processador, a partir da definição de espaço de processamento unidimensional, estabelecerá o espaço bidimensional correspondente, alterará os comandos de inicialização das variáveis `multi` para refletir o mapeamento serpentina e traduzirá os operadores de comunicação utilizados. Dessa maneira, deixaremos transparente para o programador o mapeamento proposto, permitindo que ele projete seu algoritmo para uma SIMD vetorial e explore os recursos da Zephyr de modo eficiente, sem alterar seu programa.

Referências

[1] James H. Jackson. The data transport computer: A 3-dimensional massively parallel simd computer. 1991.

[2] D. E. Knuth. *The Art of Computer Programming*, volume 3. 1973.

[3] W. R. Pearson e W. Miller. Dynamic programming algorithms for biological sequence comparison. *Methods in Enzymology*, 210:575–601, 1992.

[4] Queen. *Designing of Efficient Parallel Algorithms.* 1991.

[5] Wavetracer. *Zephyr: Installation and Operation*, April 1992.

[6] WaveTracer, Inc. *The multiC Programming Language*, September 1991.